Matti Ruupunen

PELAAJATEKOÄLYN SUUNNITTELU JA TOTEUTUS SHOOT 'EM UP - PELIIN

Opinnäytetyö
Kajaanin ammattikorkeakoulu
Tradenomi
Tietojenkäsittely
Syksy 2013

KAJAANIN AMMATTIKORKEAKOULU
UNIVERSITY OF APPLIED SCIENCES

OPINNÄYTETYÖ
TIIVISTELMÄ

Koulutusala Tradenomi	Koulutusohjelma Tietojenkäsittely
Tekijä(t) Matti Ruupunen	
Työn nimi Pelaajatekoälyn suunnittelu ja toteutus shoot 'em up - peliin	
Vaihtoehtoiset ammattiopinnot Peliohjelmointi	Ohjaaja(t) Leena Heikkinen
	Toimeksiantaja
Aika Syksy 2013	Sivumäärä ja liitteet 47

Tämän opinnäytetyön tarkoituksena on pelaajatekoälyn suunnittelu ja toteutus shoot 'em up - peliin. Pelitekoäly on ollut oleellisena osana pelattavuutta pelien alkuajoista lähtien. Pelitekoälyn toteuttamiseen on lukuisia eri tapoja. Tästä syystä tämä opinnäytetyö pyrkii selvittämään mitä kaikkea pelitekoälyn suunnittelu ja toteutus vaatii pelinkehittäjältä.

Teoriaosuuteen tutkittujen tietojen tarkoituksena on luoda pohja käytännöntyön toteutukselle. Samalla näiden tietojen avulla voidaan perustella lopullisten toteutusratkaisujen valintaa. Teoriaosuudessa käydään läpi pelitekoälyn tarkoitusta ja mitä tekoälyn suunnittelemisessa pitää ottaa huomioon. Tämän lisäksi on tutkittu useita eri tekniikoita pelitekoälyn ja tähän liittyvien ominaisuuksien toteuttamiseksi. Näihin toteutustekniikoihin kuuluvat esimerkiksi tekoälyarkkitehtuureja, reitinhakutapoja ja ohjelmointitekniikoita.

Käytännönosuuteen liittyy tekoälyn toteutuksen lisäksi myös tekoälyyn liittyvän peliprojektin toteutus. Tähän opinnäytetyöhön liittyen ei ole aikaisempaa peliprojektia, joten ennen tekoälyn toteutusta piti toteuttaa varsinainen pelikin. Käytännönosuudessa esitellään aluksi tämä peliprojekti ja toteutetun tekoälyn suunnitelma. Näiden jälkeen kuvaillaan projektin toteutusprosessi kokonaisuudessaan vaihe vaiheelta.

Opinnäytetyön lopuksi on analyysi tästä opinnäyetyöprojektista. Loppujen lopuksi aikaiseksi saatiin toimiva tekoäly, joka toimii suunnitelman mukaisesti. Tarkoittaen sitä, että tekoälyn toteutukseen hyödynnettiin teoriataustaan tutkittuja tietoja. Tämän lisäksi lopullinen tekoäly tarjoaa haastavan vastuksen, mutta tämän voittaminen ei ole mahdotonta, eikä myöskään liian helppoa. Tekoälyn toteutukseen käytetyt toteutustekniikat ovat perusteltuja ja asianmukaisia projektin tarpeisiin nähden.

Kieli	Suomi
Asiasanat	Tekoäly, reitinhaku, peliohjelmointi, pelisuunnittelu
Säilytyspaikka	☐ Verkkokirjasto Theseus ☐ Kajaanin ammattikorkeakoulun kirjasto

KAJAANIN AMMATTIKORKEAKOULU
UNIVERSITY OF APPLIED SCIENCES

THESIS ABSTRACT

School Business	Degree Programme Business Information Technology
Author(s) Matti Ruupunen	
Title Designing and Developing Player Artificial Intelligence for a Shoot 'em up Game	
Optional Professional Studies Game Programming	Instructor(s) Leena Heikkinen
	Commissioned by
Date Fall 2013	Total Number of Pages and Appendices 47

The purpose of this thesis was to design and develop a player artificial intelligence character for a shoot 'em up game. Game artificial intelligence has been a mayor part in playability since the early days of video games. There are countless ways to develop game artificial intelligence. Because of that, this thesis aims to explain what designing and developing game artificial intelligence demands from a game developer.

The purpose of the researched information in the theoretical part was to create a basis for the development of the practical part of the thesis. This information can also be used to explain the reasons for the final development choices of the practical part. The theory part explains what game artificial intelligence means and what you have to take into consideration when designing it. In addition to that, this section also describes different techniques for developing game artificial intelligence and issues related to that. For example, these techniques include game artificial intelligence architectures, pathfinding methods and programming techniques.

The practical part also includes the creation of the game project, in addition to the development of the artificial intelligence character. There is no previous project related to this thesis, so the game must be created before the character can be developed. The practical part begins with the introduction of the game project and the design plan for the player artificial intelligence character. After these the process description of the whole project is included.

The thesis ends with the analysis of the project. The produced player artificial intelligence character ended up working as planned. It means that the researched information was utilised in the development of this character. Also, the character provides adequate challenge, which means that the character is not impossible to beat, but winning is not easy either. The techniques used to create the character are appropriate for the needs of this project and the reasons for picking them have been explained in the thesis.

Language of Thesis	Finnish
Keywords	Artificial Intelligence, Pathfinding, Game programming, Game designing
Deposited at	☐ Electronic library Theseus ☐ Library of Kajaani University of Applied Sciences

SISÄLLYS

SYMBOLILUETTELO

Lua	Skriptausohjelmointikieli.
Shmup	Shoot 'em up - peligenre.
XML	Extensible Markup Language. Merkintäkieli tai standardi, joka auttaa jäsentämään tietoa selkeään muotoon.
XNA	Microsoftin kehittämä pelinkehitysympäristö.

1 JOHDANTO

Pelien alkuajoista lähtien pelitekoäly on ollut olennainen osa pelien pelattavuutta. Vuosien saatossa pelitekoäly on kehittynyt huimasti ja samanaikaisesti näiden toteutustavat ovat kehittyneet. Tämän opinnäytetyön tarkoituksena on suunnitella ja kehittää pelaajatekoäly shoot 'em up - peliin. Shoot 'em up - peligenren valintaan on muutamia syitä. Ensinnäkin pelaajatekoälyjen esiintyminen näissä peleissä on melko harvinaista. Toisekseen tämän työn ideana on luoda tekoäly, joka pystyy sopeutumaan jatkuvasti muuttuviin olosuhteisiin. Tästä johtuen tekoälyn toteuttamisessa on omat haasteensa ja shoot 'em up - peligenre soveltuu tähän erittäin hyvin.

Tekoälyjen toteuttamiseen on lukuisia eri keinoja, joten tästä syystä opinnäytetyössä selvitetään, mitä tekoälyn kehitystyö vaatii pelinkehittäjältä. Tässä työssä on tehty tutkimusta pelitekoälyn tarkoituksesta, suunnittelusta ja toteutustavoista. Näitä tutkittuja tietoja on testattu ja hyödynnetty käytännönosuuden toteutuksessa. Lopulliset toteutusratkaisut on perusteltu projektin tarpeiden ja toteutustekniikoiden tarkoituksien perusteella.

Työssä käydään läpi koko käytännönosuuden toteutusprosessi vaihe vaiheelta. Toteutusprosessin kuvauksessa selitetään esimerkiksi valittuja toteutusratkaisuja ja näissä esiintyneitä ongelmia. Toteutusta havainnollistetaan kuvien ja käytännön esimerkkien avulla. Lopulta projektin onnistuminen analysoidaan kokonaisuudessaan. Analysoinnin ideana on selvittää, kuinka projekti onnistui kokonaisuudessaan ja kuinka hyvin valitut toteutustekniikat soveltuivat tähän projektiin.

2 PELITEKOÄLY

Maailmassa on monia erityyppisiä tietokoneohjelmia, jotka hyödyntävät tekoälyä. Markkinasimulaattorit, loogiset järjestelmät ja taloussuunnittelijat ovat esimerkiksi muutamia eri tietokoneohjelmistoaloja, jotka turvautuvat tekoälyn eri elementteihin. Näitä elementtejä ovat esimerkiksi tilannelaskenta, hakupuut, ongelmanratkonta ja päätöksenteko. Yksi tietokoneteollisuusala on kuitenkin vuosien saatossa hyödyntänyt tekoälyn eri ominaisuuksia yhä enemmän ja tämä on peliteollisuus. (Wexler 2002)

2.1 Pelitekoälyn juuret

Videopelit ovat kehittyneet huimasti vuosikymmenten saatossa. Vuosi vuodelta peleistä tulee entistäkin monimutkaisempia ja kehittyneempiä teknisesti ja pelitekoäly on yksi pelien osaalue, joka kehittyy jatkuvasti huimaa vauhtia. Varhaisimmat tekoälyt peleihin olivat suunniteltu kolikkopelikoneisiin. Tekoälyn tarkoitus tällöin oli pitää huoli siitä, että pelaaja jatkoi pelaamista koneella ja syötti täten lisää kolikoita laitteisiin. Näihin peleihin kuuluivat esimerkiksi ”Pong”, ”Pac-Man”, ”Space Invaders” ja ”Donkey Kong”. (Wexler 2002, Tozour 2002)

Näiden pelien tekoäly koostui yksinkertaisista säännöistä ja toiminnoista, joihin yhdistettiin satunnaislogiikkaa. Satunnaisuus on tärkeätä tämän tyylisissä peleissä, sillä olioiden pitää vaikuttaa arvaamattomilta, jotta uudelleenpeluuarvo säilyisi. Uudelleenpeluuarvoa ylläpitää se, että pelaajan on vaikeampi ennustaa vastustajan liikkeitä. Esimerkiksi ”Pong” pelissä tietokonevastustaja teki parhaansa palauttaakseen pallon takaisin pelaajalle. Liikkumisalgoritmi oli yksinkertainen ja vaikeusasteesta riippuen vastustaja oli joko nopeampi tai hitaampi ja virhemarginaali oli samoin tavoin joko suurempi tai pienempi.(Wexler 2002, Tozour 2002)

Varhaisissa peleissä pelitekoäly ei ollut ”Pong” tekoälyä kummoisempaa, johtuen pelien yksinkertaisuudesta. Tämän ajan peleissä oli yleistä kaksinpelattavuus ihmisten välillä, tietokonevastuksien sijaan. Laitteiston tekniset rajoitukset aiheuttivat myös rajoituksia pelitekoälyn monimutkaisuuteen. Vuosien saatossa kuitenkin laitteistot ja teknologia kehittyivät huimasti ja samalla myös pelitekoäly otti askeleita eteenpäin. (Wexler 2002, Tozour 2002)

Varhaiset strategiapelit olivat pelitekoälyn kehittymisen pioneereja. Syy tähän on se, että strategiapelit vaativat hyvän tekoälyn ollakseen edes pelattavia jollain tasolla. Strategiapelitekoäly on haasteellista, sillä se vaatii hienostunutta yksikkötekoälyä, sekä myös monimutkaista strategisesti ajattelevaa tietokonevastustajatekoälyä. Varhaisimpia strategiapelihelmiä ovat esimerkiksi vuoropohjainen ”Civilization” - pelisarja ja reaaliaikainen ”Warcraft” – pelisarja. (Wexler 2002, Tozour 2002)

Peliteollisuuden kehittyessä muissakin pelityypeissä alkoi tapahtua merkittävää kehitystä tekoälyn suhteen. Joitakin pelisarjoja, jotka ovat saaneet kehuja varhaisesta tekoälystään, ovat esimerkiksi räiskintäpelisarja ”Half Life”, hiiviskelypelisarja ”Thief”, kaupunkisimulaattoripelisarja ”SimCity” ja jumalapelisarja ”Black & White”. ”Half-Life” sai kehuja taktisesta tekoälystään. ”Thief” tuli tunnetuksi tekoälyn reaktio- ja palautekyvyistä ilmaista eri tilanteita, joissa tekoäly oli. ”SimCity” puolestaan oli ensimmäisiä pelejä, jossa todistettiin potentiaalisia lähestymistapoja keinotekoiseen elämään. ”Black & White” peli eritoten on suunniteltu ja rakennettu siten, että tekoälyn monimuotoisuus nousee hyvin voimakkaasti esille. Kyseisen pelin pääideana on kouluttaa ja opettaa eräänlaista tekoälyoliota ja samalla olla tekemisissä pelimaailman asukkien kanssa. Pelaajan tekemät valinnat vaikuttavat siihen, kuinka tekoälyolio ja maailman asukkaat käyttäytyvät toisiaan ja pelaajaa kohtaan pelin edetessä. (Tozour 2002)

Nykyään laitteistot ja teknologia kehittyvät entistäkin tehokkaampaan suuntaan jatkuvasti. Tämä kehitys sallii yhä monimuotoisempien tekoälyjen kehityksen. Tämä on pelinkehittäjille jatkuva haaste, koska heidän pitää kehittää jatkuvasti monimuotoisempia tekoälyjä haastamaan pelaajia. Nykyään onkin lukemattomia erityyppisiä tekoälyjä erilaisiin tarkoituksiin, vaihtoehtoisia toteutustapoja ja käyttötarkoituksia eri tilanteisiin. Pelitekoäly voi olla hyvinkin yksinkertaista tai erittäin monimutkaista riippuen pelistä. Joten missä kulkee pelitekoälyn määrityksen rajat? (Kehoe 2013)

2.2 Mitä on pelitekoäly

Kaikessa yksinkertaisuudessaan pelitekoäly tarkoittaa tietokoneen ohjastamaa hahmoa, joka toimii pelaajan tai halutun olion kaltoin. Lähes kaikissa videopeleissä on poikkeuksetta tekoälyä jollakin tasolla. Yksinkertaisimmillaan tekoäly voi ohjastaa hahmoa, jolla on yksinkertainen liikkumiskuvio ja samalla suorittaa ennalta-arvattavia toimintoja. Monimutkaisempi te-

koäly puolestaan mahdollistaa ennalta-arvaamattoman ja tilannekohtaisen käytöksen. Toisin sanoen tekoäly voi olla sopeutuva, joka tekee päätöksiä muiden olioiden käyttäytymisen perusteella. Esimerkkinä tällaisesta tekoälystä on esimerkiksi vihollistekoäly nykyaikaisessa räiskintäpelissä. Vihollinen voi hakeutua suojaan, jos pelaaja ampuu tätä kohden. Suojasta käsin vihollinen voi joko odottaa ampumisen loppua tai sitten vaikka heittää kranaatin pelaajaa kohti. (TechTerms 2010)

Pohjimmiltaan pelitekoäly on oleellinen osa pelin pelattavuutta. Tästä syystä pelitekoälyn tärkein rooli on olla viihdyttävä, jonka takia tekoälystä suunnitellaan yleisesti ottaen epätäydellinen. Jos tekoäly on helppo voittaa, niin pelikokemuksesta tulee pelaajalle tylsempi. Samoin tavoin jos tekoäly päihittää aina pelaajan, niin tämä masentaa pelaajaa. Kummassakaan tapauksessa tämä ei ole hyvä pelin kannalta. Jotta pelikokemus on nautinnollinen, niin tekoälyn pitäisi tarjota haastava vastus, mutta samalla tämän pitäisi hävitä enemmän kuin voittaa. Tärkeintä kuitenkin on, että pelikokemus on haastava ja hauska. Tästä syystä peleissä on yleisesti ottaen vaikeusasteita, jotta pelaajat voivat valita sopivan haastavuuden itselleen, sillä pelaajien taitotaso vaihtelee erittäin paljon. (TechTerms 2010, Buckland 2005)

Pelitekoäly termi itsessään kattaa laajan valikoiman erinäisiä ohjelmointi ja suunnittelukäytänteitä. Näihin käytänteisiin kuuluvat esimerkiksi reitinhaku, tekoälyarkkitehtuurit, päätöksenteko, sääntöjärjestelmiä ja monia muita tekniikoita. Kaikki nämä käytänteet sitoo yhteen se, että pelitekoälyn tavoitteena on luoda pelaajalle mielikuva älykkäästä käyttäytymisestä. Käyttäytyminen, joka vaikuttaa tekevän asioita omien tarpeidensa mukaisesti ja vastaa pelaajan tekemiin valintoihin. (Mateas 2003)

2.3 Pelitekoälyn haasteita

Jos vertaa pelitekoälyä ja perinteistä akateemista tutkimustekoälyä keskenään, niin näiden kahden välillä on merkittävä ero. Tutkimustekoälyn tarkoituksena on löytää optimaalisia ratkaisuja oikean maailman ongelmiin. Koska tutkimustekoälyllä on tarkoituksena löytää optimaalisia ratkaisuja, niin tämä ottaa vähemmän huomioon laitteisto- tai aikarajoituksia. Tutkimustarkoituksiin tekoälyn voi jättää tunneiksi, päiviksi tai jopa viikoiksi ratkaisemaan ongelmaa. Pelitekoälyn kannalta puolestaan tämä ei ole mahdollista. (Buckland 2005)

Pelinkehittäjät joutuvat jatkuvasti kamppailemaan laitteistorajoitusten kanssa. Esimerkiksi saatavilla olevat prosessointi ja muistitehot vaihtelevat erittäin paljon eri alustojen välillä. Pelkästään tietokoneiden välillä on huomattavia eroja, konsoleista ja mobiililaitteista puhumattakaan. Laitteistorajoitukset ovatkin olleet vuosien varrella iso este tekoälyn kehityksen tiellä. Peligrafiikan piirtäminen ruudulle on perinteisesti vienyt ison osan prosessointitehosta, joten tekoälylle ei ole jäänyt paljoa aikaa, tehoja tai muistia vapaaksi. Tämän lisäksi pelikonsolien suorituskyvyn rajoittuneisuus tietokoneisiin verrattuna voi aiheuttaa suurta päänvaivaa pelinkehittäjillä. Teknologian kehittyessä tekoälylle vapautuu kuitenkin yhä enemmän prosessointitehoa. Näytönohjaimien nopeus on kehittynyt huimasti prosessorin tehoon verrattuna, joten tämä vapauttaa enemmän resursseja tekoälylle. (Buckland 2005, Tozour 2002, Nareyek 2004)

Nykypäivän pelaajilla on kuitenkin yhä enemmän kokemusta peleistä, joten he ovat tottuneet siihen, että peleistä tulee jatkuvasti monimutkaisempia. Joten pelinkehittäjillä on jatkuva haaste kehittää peleistään entistäkin mukaansatempaavampia. Jatkuvasti kehittyvä pelitekoäly on iso osa tätä kehitystä. Entistäkin monimutkaisemman ja viihdyttävämmän tekoälyn suunnittelu ja toteutus eivät olekaan yksinkertainen tehtävä. (Kehoe 2013)

3 PELITEKOÄLYN SUUNNITTELU

Tietokoneiden ohjastamien hahmojen pohjimmainen tarkoitus on luoda peliin syvyyttä ja pelattavuutta. Jotta pelaajille pystyttäisiin tarjoamaan mielenkiintoinen pelikokemus, täytyy pelinkehittäjien luoda tämänmukaisia ihmismäisiä vastustajia. Tekoälyn laatu voi vaikuttaa hyvinkin paljon pelin suosioon ja myyntiin. Oikeasti älykästä tekoälyä ei kuitenkaan ole tarkoitus toteuttaa, vaan pelaajalle tarjotaan yleisesti ottaen illuusio älykkyydestä. Kaikessa yksinkertaisuudessaan: jos pelaaja uskoo tietokone vastustajansa olevan älykäs, niin silloin se on älykäs. (Buckland 2005, Scott 2002)

Tekoälyä kehitettäessä täytyy kuitenkin olla varovainen, jotta tämä älykkyyden illuusio ei murru kesken kaiken. Jos näin pääsee tapahtumaan, niin tämä vaikuttaa suoraan pelaajan immersioon ja pelinautintoon. Tämä tapahtuu esimerkiksi silloin, jos pelaaja näkee tekoälyn käyttäytyvän typerästi. Tällaiseen käyttäytymiseen kuuluu esimerkiksi seiniä päin juokseminen, kulmiin jumiutuminen tai reagoimaton käytös tiettyjä asioita kohtaan. Tekoälyn huijaaminen on myös vakava asia, ellei pelaaja eritoten tiedä tekoälyn huijaamisesta kyseisessä pelissä tai tilanteessa. Pelinkehittäjillä onkin paljon asioita, joita pitää ottaa huomioon tekoälyä suunniteltaessa. (Buckland 2005)

3.1 Ihmismäinen käytös

Koska tietokonevastustajaa pidetään yleisesti ottaen pelaajan korvikkeena, niin tästä johtuen tälle asetetaan erilaisia odotuksia. Odotukset voivat vaihdella riippuen siitä, pelaako tietokone pelaajaa vastaan vai pelaajan kanssa. Ihmispelaajat ovat tunnettuja arvaamattomien valintojen tekemisestä. Esimerkiksi strategiapeleissä tämä voi tarkoittaa hyökkäystä isomman armeijan kimppuun hämäykseksi. Sitä vastoin ihmispelaajat osaavat käyttäytyä erittäin arvattavastikin tietyissä tilanteissa. Esimerkiksi räiskintäpeleissä pelaajat saattavat kulkea samoja reittejä tai strategiapeleissä pelaajilla on aina sama logiikka rakennusten rakentamisen suhteen alkupelissä. (Scott 2002)

Ihmispelaajat voivat siis käyttäytyä joko arvaamattomasti sekä arvattavasti. Tästä johtuen näiden kahden toiminnon välillä on selvä ristiriita, jonka takia tekoälyn kehittäminen matkimaan tätä on hyvin vaikeaa. Tekoälyllä pitäisi olla tarpeeksi satunnaisuutta uudelleenpeluuar-

von säilymisen kannalta. Samalla tekoälyn pitäisi olla myös ennalta-arvattava, jotta pelaaja pystyy päättelemään vastastrategian tietokonetta vastaan. Samaan aikaan tiekoneen pitäisi pystyä myös analysoimaan pelaajan strategiaa, jotta tämä pystyy tarjoamaan asianmukaisen vastuksen. (Scott 2002)

Arvaamattomuuden ja arvattavuuden lisäksi tekoälyllä pitäisi olla myös yllätyksellisiä ominaisuuksia. Yllätyksellisyys riippuu täysin tietenkin pelin tyypistä. Esimerkiksi strategiapelissä tämä voi tarkoittaa älykkäiden taktiikoiden toteutusta, kuten väijytyksiä, useammalta suunnalta hyökkäämistä tai häirintää. Räiskintäpelissä tämä puolestaan voisi tarkoittaa suojatulen antamista, sivustasta hyökkäämistä tai erinäisiä väijytyksiä. (Scott 2002)

Yksi oleellinen huolenaihe tekoälyn kehityksessä on myös tekoälyn huijaaminen. Useimpien ihmispelaajien mielestä tekoälyn ei pitäisi huijata, jolloin tekoälyllä olisi samat lähtökohdat kuin pelaajillakin. Pääasialliset väitteet huijaamista vastaan johtuvat siitä, että pelaajat eivät halua tietokoneella olevan epäreilua etulyöntiasemaa pelaajaa kohtaan. Tosiasia kuitenkin on, että tietokone on tasavertaisessa tilanteessa selvästikin alakynnessä, sillä tietokone ei kykene improvisoimaan. (Scott 2002)

Parhaimmillaan tekoäly kykenee valitsemaan useasta strategiasta, jotka pelinkehittäjä on tälle kehittänyt. Kun pelaaja tietää nämä strategiat, niin pelityypistä riippuen tietokone ei enää tarjoa välttämättä vastusta. Tästä syystä erilaiset huijaamisen muodot ovat olennainen osa tekoälyä, koska tekoälyn perimmäinen tarkoitus on tarjota pelaajalle asianmukaista haastetta. Jos huijaaminen ei ole ilmeistä pelaajalle, niin tällöin tämä on hyväksyttävää. Useat tekoälyn huijaamisen tavat ovatkin varsin huomaamattomia ja välttämättä niitä ei edes pidetäkään varsinaisena huijaamisena. Useasti pelinkehittäjillä ja pelaajilla on erilainen käsitys huijaamisesta. (Scott 2002)

3.2 Illuusio älykkyydestä

Ihmismäisen käytöksen matkimisen lisäksi on myös muitakin tapoja saada tekoäly näyttämään älykkäältä. Ennalta määritetyt eli ”skriptatut” toiminnot ovat yksi osa-alue, joiden avulla tekoälystä saa hyvinkin älykkäänoloisen pelaajan silmiin. Tämä tarkoittaa sitä, miten tekoäly toimii esimerkiksi ennen ja jälkeen pelaajaan törmäämisen. Muutamia tällaisia toimintoja on vaikka se, että räiskintäpeleissä vartijat keskustelevat keskenään, ennen kuin huomaavat

pelaajan ja taistelu alkaa. Jopa pelkästään vartioreittien lisääminen näille saa tekoälyn näyttämään paljonkin älykkäämmältä. Silläkin on suuri ero, onko vihollinen valmiiksi taisteluasemissa vai laskeutuvatko nämä esimerkiksi seiniä pitkin taistelualueelle. (Hale 2007)

Ennalta määriteltyjen toimintojen lisäksi tekoälyn pitäisi myös ilmaista tekemisiään pelaajalle tavalla tai toisella. Tekoäly, joka ei ilmaise millään tavalla ennakkoon tekemisiään, vaikuttaa pelaajan silmiin typerältä. Esimerkiksi räiskintäpeleissä viholliset saattavat huutaa ”KRANAATTI!”, ennen kuin heittävät tämän ja samalla hahmo animoituu asian mukaisesti. (Hale 2007)

Tekoälyn pitäisi näyttää siltä, että tällä on tavoitteita, joita tämän pitäisi toteuttaa. Jos pelaaja ei ymmärrä tekoälyn toimintoja, niin tekoälyn käytös vaikuttaa järjettömältä. Jos vihollinen yhtäkkiä juoksee suojasta pois, niin tämä vaikuttaa yksinään typerältä käyttäytymiseltä pelaajalle. Tosin jos samaan aikaan toinen vihollinen huutaa käskyjä kyseiselle viholliselle, niin tämä saa tekoälyn käyttäytymisen näyttämään tarkoituksenmukaiselta. Tärkein asia kuitenkin on se, että tekoälyn pitäisi kyetä ilmaisemaan itseään tavalla tai toisella, joko verbaalisesti tai muuten hahmoanimaatioiden kautta. Tämä tuo paljon syvyyttä tekoälyn toimintaan ja saa tämän näyttämään älykkäämmältä kuin tämä oikeasti onkaan. (Hale 2007)

Yksi tärkeä asia tekoälyissä on näiden elinikä. Tekoäly voi olla erittäin monimutkainen ja älykäs, mutta tällä ei ole merkitystä, jos pelaaja päihittää nämä ennenaikaisesti. Jos pelaaja päihittää vihollisia liian nopeasti, niin näillä ei ole aikaa näyttää monimutkaisia toimintojaan. Esimerkiksi ”Halo”-pelisarjassa testattiin ihmisillä useissa eri testitapahtumissa tekoälyjä, joiden kestävyys vaihteli. Tätä ei tietenkään kerrottu itse testihenkilöille. Näissä testeissä tekoäly toimi samalla tavalla, mutta kestävyyspisteet vaihtelivat heikompaan ja vahvempaan suuntaan. Pelaajien mielikuva näiden tekoälyjen vaativuudesta ja älykkyydestä vaihtelivat erittäin paljon näiden versioiden välillä, vaikkakin tekoälyjen käytös oli samanlainen. (Buckland 2005, Hale 2007)

Kestävyyden lisääminen tekoälylle ei kuitenkaan ole ainoa tapa lisätä näiden elinikää. Tekoälylle voi lisätä kykyjä, jotka auttavat selviytymään pidempään tai pelaajan toimintoja voi säätää siten, että vastustajia on vaikeampi päihittää. Sekin auttaa asiaa, jos pelaaja taistelee vihollisryhmiä vastaan kerrallaan. Tällöin yksittäinen vihollinen voi olla heikko, mutta koska vihollisia on useampi, niin nämä vaikuttavat älykkäämmiltä kuin nämä oikeasti ovatkaan. (Buckland 2005, Hale 2007)

Pelaajien ennakko-odotusten muovaaminen on myös olennainen osa tekoälyä. Jos pelaaja kamppailee ihmisen näköistä tekoälyä vastaan, niin pelaaja odottaa tämän käyttäytyvän kuin ihminen. Tämä on siinä mielessä vaikea saada toteutettua oikein, koska pelaajilla on erilaisia mielikuvia oikeanlaisista toiminnoista. Näihin ennakko-odotuksiin vaikuttavat pelaajien omat henkilökohtaiset kokemukset asioista ja nämä voivat vaihdella erittäin paljon. (Hale 2007)

Tästä syystä esimerkiksi avaruusoliot, robotit tai muut fantasiaoliot ovatkin paljon helpompia saada näyttämään älykkäämmiltä pelaajan silmiin. Tämä johtuu siitä, että pelaajilla on pienemmät odotukset näitä olioita kohtaan, sillä pelaajilla ei ole käytännön kokemuksia näistä, muuten kuin elokuvista ja muista peleistä. Kukaan ei tiedä, kuinka tuntemattoman olion on tarkoitus käyttäytyä, joten pelinkehittäjillä on näiden suhteen vapaat kädet keksiä toimintatapoja. Epäihmismäisten olioiden käyttäminen antaakin paljon varaa tekoälyn suunnittelun suhteen. (Hale 2007)

Pohjimmiltaan tärkein asia tekoälyn suhteen on kuitenkin se, että tämän on tarkoitus olla hauska ja haastava. Tämän tavoitteen saavuttamiseen on lukuisia eri tapoja. Tekoälyn tarkemmat tekniset toiminnot riippuvat täysin pelin tyypistä ja tekoälyolion tarkoituksesta, joten ainoaa oikeaa tapaa luoda tekoälyjä ei ole olemassa. Tärkeintä kuitenkin on, että tekoälyyn on yhdistetty pelin kannalta oikeat mekaniikat ja käyttäytymiset, jotta pelaaja kokee tämän viihdyttäväksi ja älykkääksi. Useimmiten tekoäly itsessään on erittäin yksinkertainen, mutta se on saatu näyttämään älykkäältä. Jos tarjolla on yksinkertainen oikealla tavalla toimiva tekoäly ja monimutkainen virheellisesti toimiva tekoäly, niin yleisesti ottaen yksinkertaisempi ratkaisu on aina parempi vaihtoehto. (Scott 2002, Hale 2007)

4 PELITEKOÄLYARKKITEHTUUREJA

Perimmäinen idea tekoälyn takana on sen päätöksentekokyky. Tämä on kyky, joka määrittelee, miten tekoäly toimii missäkin tilanteessa. Jotta tekoäly kykenee tekemään päätöksiä, tarvitsee se älykkään järjestelmän hallitsemaan näitä valintoja. Useimmiten pelit käyttävät yksinkertaisia arkkitehtuureja kuten tilakoneita (State machine) tai erilaisia puujärjestelmiä (Decision tree, Behavior tree). Myös monimutkaisempia arkkitehtuureja esiintyy, kuten sumeaa logiikkaa (Fuzzy logic) ja neuroverkkoja (Neural network). Nämä monimutkaisemmat arkkitehtuurit mahdollistavat älykkäämpiä tekoälyjä, mutta näitä on kuitenkin vaikeampi saada asiallisesti toimimaan verrattuna yksinkertaisempiin vaihtoehtoihin. (Millington 2006, Kehoe 2013)

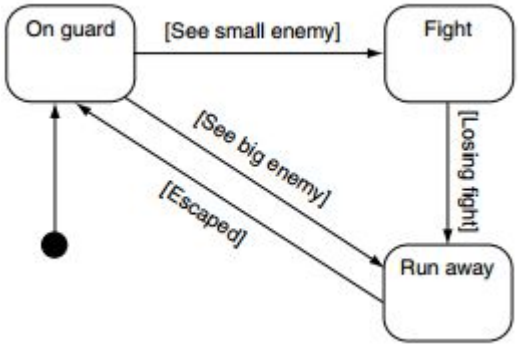
Vaikkakin olemassa olevia tekoälyarkkitehtuureja on lukuisia, niin pohjimmiltaan kaikki toimivat saman periaatteen mukaisesti. Arkkitehtuurista riippumatta tekoälyllä on aina hallussa samat tiedot, joiden pohjalta tekoäly tekee päätöksiä. Arkkitehtuurien eroina ovat menetelmät, joilla tätä tietoa hyödynnetään. Tekoälyllä on aina jokin toimintatavoite ja arkkitehtuurit auttavat pääsemään tähän tavoitteeseen. (Millington 2006)

4.1 Äärellinen tilakone

Äärelliset tilakoneet (Finite state machine) ovat vanhin ja yleisin tapa toteuttaa tekoälyarkkitehtuureja. Näitä on käytetty pelien alkuajoista lähtien. Vaikka muut arkkitehtuurit ovatkin yleistyneet, niin äärelliset tilakoneet ovat pysyneet yleisimpinä vaihtoehtoina. (Buckland 2005)

Äärellisen tilakoneen ideana on jakaa tekoälyn käyttäytyminen helposti hallittaviin osiin tai tiloihin. Näillä tiloilla on siirtymäehtoja, joilla siirrytään toisiin tiloihin. Eräs yksinkertainen esimerkki äärellisestä tilakoneesta on esimerkiksi valokatkaisin. Katkaisin voi olla kahdessa eri tilassa, joko tämä on päällä tai pois päältä. Kun sormella painaa katkaisinta, niin siirtymäehto täyttyy ja katkaisin menee joko päälle tai pois päältä, riippuen aikaisemmasta tilasta. Tietokonetekoälyn käyttäytyminen on kuitenkin monimutkaisempaa, jonka takia tiloja ja siirtymäehtoja voi olla muutamista useisiin kymmeniin. (Buckland 2005)

Äärellisessä tilakoneessa (Kuvio 1.) tekoäly voi olla vain yhdessä tilassa kerrallaan. Tämä tarkoittaa sitä, että niin pitkään kuin tekoäly on yhdessä tilassa, niin tämä tekee kyseisen tilan mukaisia toimintoja. Tilat ovat yhteydessä toisiinsa erilaisten siirtymäehtojen kautta. Jos tekoäly päättää, että siirtymäehdot ovat täyttyneet, niin tällöin tekoäly vaihtaa nykyisen tilansa kyseisen siirtymän mukaiseen tilaan. Kun tekoäly on uudessa tilassaan, niin tällöin tämä tekee kyseisen tilan mukaisia toimintoja kunnes tämän tilan siirtymäehdot täyttyvät ja tekoäly vaihtaa uudelleen tilaansa johonkin toiseen. Tämä logiikka kierre jatkuu niin pitkään kun tekoäly on elossa pelimaailmassa. (Millington 2006)



Kuvio 1. Yksinkertainen tilakone. (Millington 2006, 319)

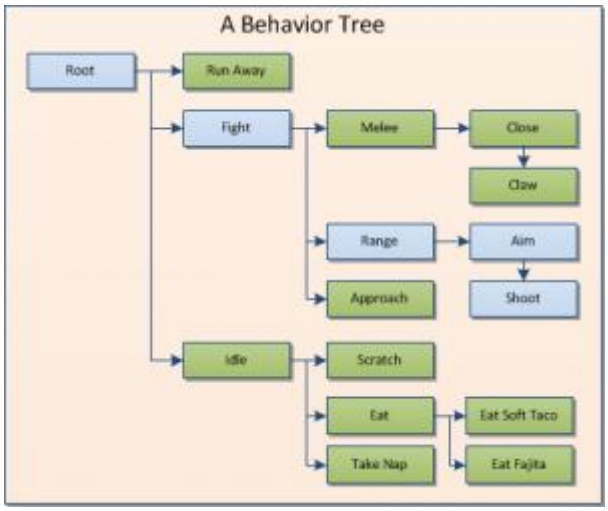
Äärellisten tilakoneiden suosiolle on useita syitä. Ensinnäkin tämän toteuttaminen on nopeaa ja yksinkertaista. Äärellisen tilakoneen toteuttamiseen on lukuisia eri tapoja ja kaikki ovat suhteellisen yksinkertaisia periaatteeltaan. Tästä johtuen näiden toteuttamiseen ei ole varsinaista oikeaa tapaa, joten täysin samanlaisia tilakoneita harvemmin näkee eri pelikehittäjien välillä. (Buckland 2005, Millington 2006)

Tilakoneet ovat myös erittäin joustavia. Yksinkertaisen toteutuksensa lisäksi niitä pystyy skaalaamaan erittäin helposti. Jos tekoälylle pitää lisätä uusia käyttäytymismalleja, niin pelinkehittäjän tarvitsee vain lisätä uusia tiloja ja siirtymäehtoja tälle. Koska tekoäly on jaettu useaan pienempään tilaan, niin näiden muokkaaminen on selkeämpää. Tämä mahdollistaa myös nopean virheenetsinnän. Jos tekoälyssä ilmenee ongelmia, niin virheen voi jäljittää juuri oikeaan tilaan, jossa tämä ilmenee. Äärelliset tilakoneet ovat myös kevyitä prosessointitehon kannalta. Tämä johtuu tilakoneiden valmiiksi määritellyistä säännöistä, joten tekoälyn ei tarvitse tehdä varsinaista oikeaa ajattelua ollenkaan. (Buckland 2005)

4.2 Käytöspuurakenne

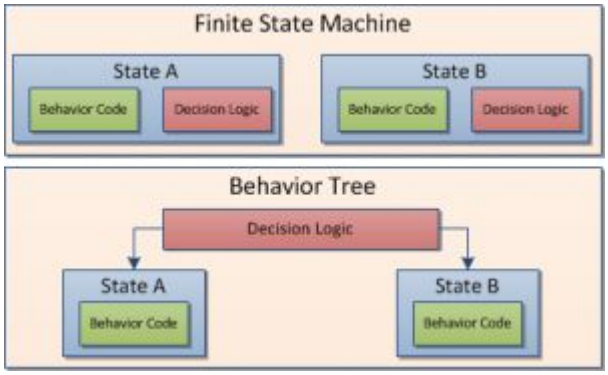
Muita järjestelmällisiä päätöksentekoarkkitehtuureja ovat erilaiset puurakenteet. Yksi yleisesti käytetty puurakenne on käytöspuurakenne (Behavior tree). Käytöspuurakenteet voivat olla hiukan rajoittuneempia kuin tilakoneet, mutta puurakenteet mahdollistavat järjestelmällisemmän lähestymistavan tekoälyn toimintaan. (Knafla 2011)

Käytöspuurakenteen käyttäytymislogiikka koostuu hierarkkisesti järjestetyistä alipuista (Kuvio 2). Nämä alipuut koostuvat solmuista, jotka kuvastavat tiloja tai siirtymäehtoja eri tiloille. Käytöspuun toiminta alkaa aina puun juuresta. Tällöin tekoäly käy alipuita läpi yksi kerrallaan, kunnes tämä löytää halutun toiminnon. Kuten tilakoneillakin, käytöspuilla on lukuisia eri toteutustapoja, joten ainutta oikeaa toteutustapaa ei ole olemassa. (Knafla 2011)



Kuvio 2. Yksinkertainen käytöspuurakenne (Mark 2012)

Äärellisen tilakoneen ja käytöspuurakenteen välillä on yksi oleellinen ero. Tilakonejärjestelmissä tekoäly voi olla vain yhdessä tilassa kerrallaan ja tämä tekee kyseisen tilan mukaisia toimintoja. Jokainen tila sisältää myös omat päätöksenteko- ja siirtymäehtonsa, joiden avulla tilaa vaihdetaan. Tämä siirtymälogiikka ei kuitenkaan varsinaisesti liity kovinkaan paljon kyseisen tilan käyttäytymiseen. Sen sijaan siirtymäehdot liittyvät enemmänkin siihen mitä tekoälyhahmon ulkopuolella tapahtuu. Käytöspuurakenteissa asia on kuitenkin toisin, sillä näissä tilan käyttäytyminen ja siirtymiin liittyvä päätöksentekologiikka on eroteltu toisistaan. (Mark 2012)



Kuvio 3. Tilakoneen ja käytöspuun ero (Mark 2012)

Tämä käyttäytymisen ja päätöslogiikan erottelu (Kuvio 3) on käytöspuurakenteiden vahvuus. Kummatkin asiat ovat olemassa, mutta päätäntälogiikka on erillinen arkkitehtuurinsa sen sijaan, että jokaisella tilalla olisi omat päätöslogiikat. Suurin etu tällaisessa järjestelmässä on se, että kaikki päätöslogiikkaan liittyvät asiat sijaitsevat yhdessä paikassa. Tästä logiikkajärjestelmästä voi tehdä niin monimutkaisen kuin vain haluaa ilman, että pitäisi jatkuvasti pitää huolta erillisten tilojen synkronoinnista keskenään. Jos uusia tiloja lisätään, niin siirtymälogiikka tarvitsee tehdä vain yhteen paikkaan. Toisin kuin tilakoneissa, joissa jokaista tähän liittyvää tilaa pitää käydä erikseen muokkaamassa. (Mark 2012)

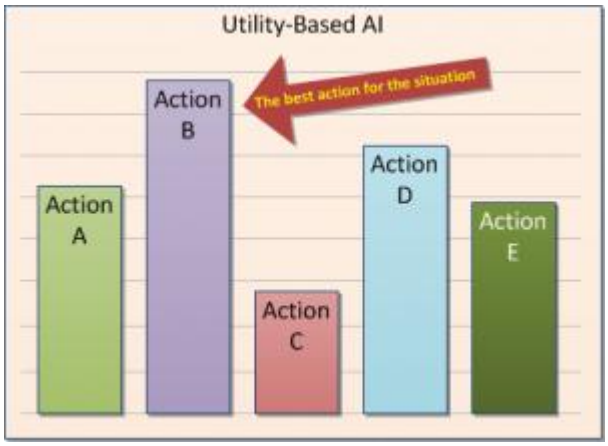
Tilojen monimuotoisemmat rakentamismahdollisuudet ovat myös yksi suuri etu käytöspuurakenteissa. On olemassa erilaisia työkaluja, pohjia ja rakenteita, joilla pystyy toteuttamaan hyvinkin yksityiskohtaisia järjestelmiä. Käytöspuurakenteen ideana on, että samoihin tarkoituksiin sopeutuvia tiloja pystyy ryhmittelemään toisiinsa sen sijaan, että on olemassa monia yksittäisiä tiloja. (Kuvio 2). (Mark 2012)

4.3 Hyödyllisyyteen pohjautuva päätöksenteko

Erinäisten tilakoneiden ja puurakenteiden lisäksi on myös olemassa vähemmän järjestelmällisiä tapoja toteuttaa tekoälyarkkitehtuureja. Yksi näistä on hyödyllisyyteen pohjautuva päätöksenteko (Utility-based system).

Tällä järjestelmällä ei ole ennalta määriteltyjä toimintamalleja sille, mitä missäkin tilanteessa on tarkoitus tehdä. Sen sijaan tämän järjestelmän ideana on antaa mahdollisille toiminnoille hyötyarvo. Hyötypistemäärä määritellään kyseisen toiminnon hyötyjen ja haittojen perusteel-

la. Kun kaikilla mahdollisilla toiminnoilla on arvo, niin tekoäly valitsee toiminnon, jolla on eniten pisteitä (Kuvio 4). (Mark 2012)



Kuvio 4. Hyödyllisyyteen pohjautuva päätöksenteko (Mark 2012)

Tällaista järjestelmää voidaan käyttää useissa erityyppisissä peleissä. Asianmukaisin käyttötarkoitus on kuitenkin silloin, kun tekoälyllä on useita eri toimintavaihtoehtoja, joista mikään ei ole välttämättä juuri oikea vastaus. Tästä syystä tekoälyllä pitää olla kyky arvioida vaihtoehdot ja valita asianmukainen ratkaisu ongelmaan. (Mark 2012)

Toisaalta tällaisessa järjestelmässä on yksi haittapuolikin, sillä pelin tapahtumia on vaikeata ennustaa etukäteen. Päätöksenteon seurauksena tekoäly valitsee sillä hetkellä asianmukaisimman vaihtoehdon, mutta pelitilanne saattaa saada odottamattomia käänteitä. Tästä syystä päätöksenteon valinnat eivät välttämättä olekaan oikeita, vaikka hetki sitten asia siltä saattoikin näyttää. Sääntöpohjaiset järjestelmät, kuten tilakoneet, käyttäytyvät aina samalla tavalla tietyissä tilanteissa. Sen sijaan hyötypohjainen päätöksenteko pohjimmiltaan tarjoaa vain ehdotuksia siitä, että miten tekoälyn pitäisi käyttäytyä kyseisessä tilanteessa. (Mark 2012)

Hyötypohjaisessa päätöksenteossa ja tilakoneissa on kuitenkin samankaltaisuuksiakin. Kun päätös toiminnoista on tehty, niin tekoälyn pitää silti siirtyä kyseiseen tilaan ja toimia päätöksen mukaisesti kunnes on aika valita uudestaan. Hyötypohjaisessa päätöksenteossa on olemassa kaikki samat tilat kuin tilakoneissakin, mutta näiden välinen siirtymä on vapaamuotoisempaa. Tästä syystä toimintojen lisääminen ja muokkaus on helppoa, sillä ne on jaettu helposti hallittaviin osiin. (Mark 2012)

4.4 Oikean arkkitehtuurin valinta

Koska eri tekoälyn toteutusarkkitehtuureja on lukuisia erilaisia, niin oikean tyypin valinta saattaa olla visainen pulma. Oikean arkkitehtuurin valinnan kysymys nouseekin useasti esille pelinkehittäjien kesken. Tosiasia on kuitenkin se, ettei ole olemassa yhtä ainoaa oikeaa tapaa, jolla tekoälyn toteuttaisi. (Mark 2012)

Oikean arkkitehtuurin valinta riippuu useista eri asioista. Kuten esimerkiksi, mihin tarkoitukseen tekoäly on? Onko teknisiä rajoituksia? Miten kokeneita pelinkehittäjät ovat? Minkälainen on aikataulu? Kuinka helposti muokattava tekoälyn täytyy olla? Nämä ovat muutamia kysymyksiä, joita pitää ottaa huomioon oikeaa arkkitehtuuria valittaessa. Millä tahansa arkkitehtuurilla saa tekoälyn toteutettua, mutta mikä tahansa vaihtoehto ei välttämättä ole kuitenkaan asianmukainen kyseiseen tarkoitukseen. Asiattoman vaihtoehdon valitseminen saattaakin aiheuttaa lisää ongelmia pelinkehittäjille ajan kuluessa. (Mark 2012)

Pohjimmiltaan jokainen tekoälyarkkitehtuuri suorittaa kuitenkin saman asian, mutta näiden mekaniikat ja toimintatavat vaihtelevat. Jokaisella tekoälyarkkitehtuurilla on omat vahvuutensa ja heikkoutensa. Yhtä oikeaa toteutustapaa ei ole, joten oikean arkkitehtuurin valinta jää pelinkehittäjien harteille heidän tarpeidensa mukaisesti. Tästä syystä pelinkehittäjien pitää tutustua eri arkkitehtuureihin ja siltä pohjalta valita se oikea juuri kyseisiin tarpeisiin. Joissakin tapauksissa pelissä saatetaan käyttää useampaa eri arkkitehtuuria tai muutaman arkkitehtuurin yhdistelmiä. Kokeilun kautta oikean vaihtoehdon löytää varmasti. (Mark 2012)

5 REITINHAKU OSANA PELITEKOÄLYÄ

Päätöksenteon lisäksi yksi oleellinen osa-alue tekoälyä on tämän reitinhakukyky. Lähes poikkeuksetta tekoälyllä on tarve liikkua pelimaailmassa. Joskus nämä liikkumismetodit voivat olla yksinkertaisia ja valmiiksi toteutettuja. Valmiiden liikkumismallien toteuttaminen on yksinkertaista, mutta pelityypistä riippuen tämän käyttäytymisessä saattaa esiintyä ongelmia. Tällainen liikkumistapa voi jäädä hyvin helposti jumiin pelikentän elementteihin, kuten esimerkiksi jos tekoälyn tielle tulee yhtäkkiä esteitä. Hyvä reitinhaku on olennainen osa tekoälyn älykkyyden illuusiota. (Millington 2006)

Reitinhaun pohjimmainen tarkoitus on löytää lyhin mahdollinen reitti pisteestä A pisteeseen B nopeasti ja tehokkaasti. Jokaisella tekoälyn liikkeellä on aloitus- ja päätöspiste. Pelityypistä riippuen ei kuitenkaan aina etsitä välttämättä lyhintä reittiä, vaan esimerkiksi turvallisinta vaihtoehtoa. Oleellisin osa reitinhakujärjestelmää kuitenkin on, että haku on tehokasta ja optimoitua. Reitinhaku voi viedä erittäin paljon prosessointitehoja, varsinkin jos tekoäly hakee reittiä, jota ei ole olekaan olemassa. (Millington 2006, Graham 2003)

Reitinhakuun käytetäänkin yleisesti ottaen ennalta määriteltyjä tietorakenteita. Yksi yleisin tapa on jakaa pelikenttä ruudukkoihin, joiden avulla tekoäly etsii reittejä. Yleisiä ruudukkoja hyödyntäviä järjestelmiä ovat esimerkiksi A* ja Dijkstra. Muita vaihtoehtoisia tapoja toteuttaa reitinhakuja on esimerkiksi käyttää navigointikarttoja tai välietappipisteitä, joita tekoäly voi hyödyntää. Reitinhakuun on erittäin paljon erilaisia toteutustapoja, jotka pätevät eri tilanteisiin. Tästä syystä reitinhaun toteutukseen ei ole yhtä oikeaa tapaa, vaan oikea järjestelmä pitää valita tarpeiden mukaisesti. On myös mahdollista yhdistellä eri tapoja toisiinsa. (Buckland 2005, Graham 2003)

5.1 A*

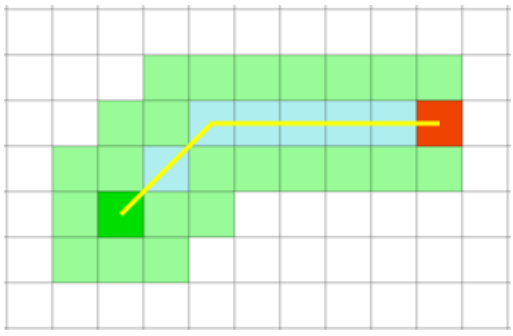
Yksi yleisesti käytetty reitinhakutapa on ruudukkopohjainen A*-metodi. Pohjimmaltaan A* on yksinkertainen toteuttaa ja erittäin tehokas reitinhakutapa. A*-metodilla on myös paljon varaa muunnelmiin tai optimointiin. Tästä metodista voi tehdä hyvinkin monimutkaisen tai yksinkertaisen tarpeiden mukaisesti. (Millington 2006)

Avainkäsitteitä A*-metodissa ovat listat ruuduista ja kustannustehokkain tai lyhin reitti lopetuspisteeseen. Teoriatasolla A*-metodi toimii siten, että reittiä lähdetään etsimään lyhimmän reitin periaatteella. Tämä tarkoittaa sitä, että aloituspisteestä lähdetään suoraan kohti päätöspistettä. Tämän navigoinnin kautta etsitään ja rakennetaan lopullinen reitti. (Lester 2005)

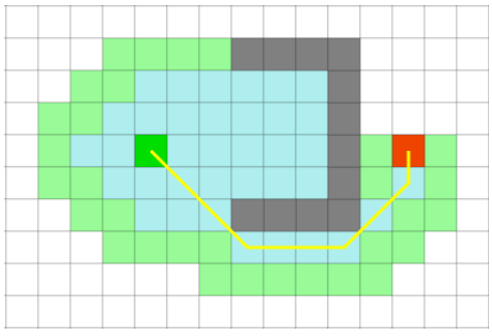
Käytännössä reitinhaku lähtee liikkeelle aloitusruudusta. Aloitusruudun ympäriltä poimitaan kaikki ruudut, joilla hahmo voi liikkua. Näille ruuduille arvioidaan arvo, jonka pohjalta metodi jatkaa toimintaansa. Tämä arvo voi koostua esimerkiksi päätöspisteen etäisyyden ja kulkemiskustannuksen summasta. Kun arvot on laskettu, niin metodi siirtyy siihen ruutuun, jolla on pienin laskettu arvo. Tämä tarkoittaa sitä, että kyseinen ruutu on joko lähimpänä päätöspistettä tai on muuten kustannustehokkain läpikäydyistä ruuduista. (Lester 2005)

Jokainen läpikäyty ruutu lisätään avoimeen listaan, jota hyödynnetään pitkin reitinhaku prosessia. Tämän lisäksi jokaisesta ruudusta tallennetaan samalla tieto ruudusta, jonka vieressä se sijaitsi. Jokainen ruutu, jossa reitinhaku on käynyt, lisätään puolestaan suljettuun listaan, sillä näitä ruutuja ei tarvitse enää käsitellä reittiä etsiessä. (Lester 2005)

Kun metodi on siirtynyt seuraavaan ruutuun, niin tarkastetaan jälleen tämän ruudun ympärillä olevat ruudut. Tässä vaiheessa jo läpikäytyjä ruutuja ei tarkisteta enää uudelleen. Metodi jatkaa toimintaansa tällä tavalla, niin pitkään kunnes päätöspiste on saavutettu (Kuvio 5) tai reitinhaku on päätynyt umpikujaan (Kuvio 6). (Lester 2005)



Kuvio 5. A* ja esteetön ympäristö (Qiao)



Kuvio 6. A* ja esteellinen ympäristö (Qiao)

Jos reitinhaku on umpikujassa, niin tällöin hyödynnetään avointa ruudukkolistaa, jota on täydennetty pitkin prosessia. Tässä vaiheessa lista lajitellaan näiden laskettujen arvojen perusteella ja näistä valitaan se ruutu, jolla on pienin arvo. Reitinhakua lähdetään jatkamaan uudelleen valitusta ruudusta ja prosessia jatketaan kunnes päätöspiste löydetään tai reitinhaku on

uudestaan umpikujassa. Tämä prosessikierre jatkuu niin pitkään kunnes reitti on löydetty tai todetaan, että reittiä ei ole olemassa. (Lester 2005)

Kun päätöspiste on löydetty, niin tällöin alkaa reitin rakennus takaisin lähtöpisteeseen. Tässä reitin rakennuksessa hyödynnetään ruutujen välisiä yhteyksiä, joita on tallennettu pitkin reitinhakuprosessia suljettuun listaan. Joten tässä vaiheessa lähdetään liikkeelle päätösruudusta. Tästä ruudusta katsotaan, mikä ruutu on yhteydessä tähän ja tämän jälkeen siirrytään kyseiseen ruutuun. Tämä ruudusta toiseen siirtyminen jatkuu niin pitkään kunnes reitti on rakennettu takaisin alkupisteeseen. Tämän tuloksena pitäisi olla lyhyin tai kustannustehokkain reitti kyseisten alku- ja päätöspisteiden välillä, riippumatta siitä miten iso alue on tutkittu tätä reittiä laskettaessa. (Lester 2005)

A*-metodi toimii erityisen hyvin esteettömässä (Kuvio 5) ympäristössä johtuen siitä, että reitinhaku etenee kohti päätöspistettä. Jos pelikentällä on esteitä (Kuvio 6), niin metodi päätyy tutkimaan paljon isomman alueen kuin loppujen lopuksi on tarvetta. Reitinhaku ei tosin tätä voi tietää ennakkoon, mutta lopulta syntyy kuitenkin optimaalisin reitti pisteiden välillä etsityn alueen koosta huolimatta. A* toimiikin parhaiten tilanteissa, joissa on selkeä aloitus- ja päätöspiste, sekä pelikenttä ei sisällä tuntemattomia alueita. (Lester 2005)

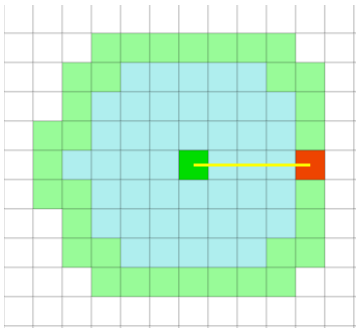
5.2 Dijkstra

Vaikka A*-metodia saatetaan pitää yleisimpänä ja ehkä jopa parhaimpana reitinhakumetodina, niin tämä ei kuitenkaan sovellu kaikenlaiseen toimintaan. Eräs toinen reitinhaku metodi on Dijkstra, joka on myös ruudukkopohjainen ja hyvin samankaltainen kuin A*-metodi. (Lester 2005)

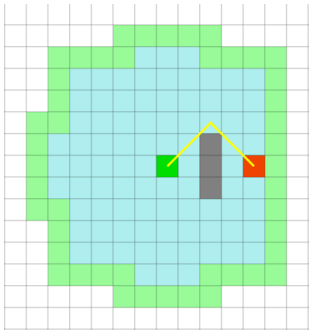
Pääasiallinen ero A*-metodin ja Dijkstran välillä on se, että Dikstrassa ei lasketa ruuduille hyötyarvoja. Tämä tarkoittaa sitä, että Dijkstrassa reitinhakualue laajenee joka suuntaan jatkuvasti (Kuvio 7, Kuvio 8). Käytännössä Dijkstra lähtee liikkeelle aloitusruudusta ja tarkistaa tämän ympäröimät kulkukelpoiset ruudut. Kaikki läpikäydyt ruudut lisätään avoimeen listaan, jonka jälkeen reitinhakua laajennetaan eteenpäin näistä läpikäydyistä ruuduista. (Lester 2005)

Kaikki ruudut, jonka ympäröimät ruudut on tarkastettu, lisätään suljettuun listaan. Suljettu lista sisältää ruudut, joita ei tarvitse enää tarkistaa uudelleen reitinhaun aikana. Reitinhaku jatkuu siten, että kaikki avoimen listan ruudut käydään läpi ja näiden ympäröimät läpikäymättömät ruudut lisätään jälleen avoimeen listaan. Tämä kierre jatkuu niin pitkään kunnes päätöspiste on lopulta saavutettu tai havaitaan, että reittiä ei ole olemassa. (Lester 2005)

Kun päätöspiste on löydetty, niin reitti rakennetaan samalla periaatteella kuin A*-metodissa. Käytännössä tämä tarkoittaa sitä, että ruuduista tarkistetaan tiedot näiden yhteyksissä olevista ruuduista ja näiden tietojen pohjalta rakennetaan optimaalisin reitti aloitus- ja päätöspisteen välille. (Lester 2005)



Kuvio 7. Dijkstra ja esteetön ympäristö (Qiao)



Kuvio 8. Dijkstra ja esteellinen ympäristö (Qiao)

Etsintäalueen laajuus tekee Dijkstrasta huomattavasti hitaamman kuin A*-metodista. Dijkstralla on kuitenkin omat käyttötarkoituksensa, johon A* ei sovellu kovinkaan hyvin. Dijkstra toimii erityisen hyvin tilanteissa, joissa on joko tuntematon maasto tai useita päätöspisteitä. Esimerkiksi ruutupohjaisissa strategiapeleissä käytetään erittäin paljon Dijkstra-metodia määrittelemään yksiköiden liikealueetta. (Lester 2005)

5.3 Dynaaminen reitinhaku

Dynaaminen reitinhaku on enemmänkin ajattelumalli, jota sovelletaan olemassa oleviin reitinhakumetodeihin, kuin itsenäinen reitinhakutapa. Joissakin peleissä pelikentällä ei tapahdu sen isompia muutoksia ja voidaan olettaa, että kaikki liikkumiskustannukset pysyvät samoina koko pelikerran lävitse. Tällöin voidaan käyttää perinteisiä reitinhakumetodeja, mutta mitä tapahtuukaan, jos ympäristö muuttuu jatkuvasti odottamattomalla tavalla tai tieto on muuten epätäydellistä. (Millington 2006)

Yllättävissä tilanteissa edelleenkin toimivat normaalit reitinhakumetodit. Kun tekoäly saa tietoonsa muutoksia pelikentällä, niin tämän tarvitsee vain suunnitella uudestaan reittivalintansa uusien tietojen pohjalta. Tämä toimii useissa tapauksissa tarpeeksi hyvin, mutta ongelmaksi syntyy se, jos pelikenttä muuttuu jatkuvasti. Tällaisessa tilanteessa normaali reitinhakumetodi ennemmin tai myöhemmin tukkeutuu. Tarkoittaen sitä, että reittejä lasketaan uudelleen ennen kuin edellistä on saatu valmiiksi. Tällöin tekoälyllä ei tapahdu reitinhaun suhteen minkäänlaista edistystä. (Millington 2006)

Tästä syystä dynaaminen reitinhaku on enemmänkin ajattelumalli, jota sovelletaan olemassa oleviin reitinhakumetodeihin. Dynaamisen reitinhaun ideana on se, että tekoälyn tarvitsee laskea vain muuttuneet osat uudelleen. Koska dynaaminen reitinhaku on ajattelumalli, niin tästä syystä tämän toteutustavat voivat vaihdella erittäin paljon. On myös virallisia tapojakin, kuten esimerkiksi A*-metodin dynaamista versiota kutsutaan D*-metodiksi. (Millington 2006)

Dynaamiset muunnelmat vähentävät yleisesti ottaen reitinhaun suoritusaikaa muuttuvissa ympäristöissä, mutta toisaalta nämä aiheuttavat enemmän prosessointiteho rasitusta. Tehokustannukset johtuvat yksinkertaisesti siitä, että reittejä joudutaan laskemaan enemmän ja dynaamisen version reitinhakumetodin laskenta on raskaampaa kuin normaalin version. (Millington 2006)

6 PELITEKOÄLYN OHJELMOINTITEKNIIKOITA

Vuosien saatossa siinä missä teknologia on kehittynyt eteenpäin, niin tekoälyn ohjelmoimiseen käytettävät ohjelmointikieletkin ovat kehittyneet. Pitkään on ollut tapana kirjoittaa tekoälyyn liittyvä koodi suoraan projektin lähdekoodiin. Tämä ratkaisu on nopeaa toteuttaa ja se toimii vieläkin pienemmissä ja yksinkertaisissa projekteissa. (Millington 2006, Buckland 2005)

Nykyään kuitenkin isommat peliprojektit ovat kehittyneet entistä monimutkaisemmaksi ja samalla myös lähdekoodin pituus on kasvanut samassa suhteessa. Tästä syystä lähdekoodin kääntäminen saattaa viedä pitkiä aikoja, joka on epäkäytännöllistä, jos tarvitsee muuttaa vain muutamia pieniä arvoja tai asioita. Tämän seurauksena on kehitetty tapoja erottaa sisältöä lähdekoodista erillisiin tiedostoihin. (Millington 2006, Buckland 2005)

6.1 Skriptaus

Yksinkertaisimmillaan skriptaus tarkoittaa sitä, että pelimoottori hyödyntää pelikoodissa ulkopuolisia tiedostoja. Skriptaukseksi voidaan luokitella esimerkiksi ulkopuoliset asetustiedostot, joista pelin koodi poimii tietoa. Tämä mahdollistaa sen, että lähdekoodia ei tarvitse aina kääntää uudestaan, vaan sen sijaan pelimoottoriin luodaan koodia, joka lukee ja tulkitsee näitä ulkopuolisia tiedostoja. Jos on tarvetta muuttaa arvoja, niin tarvitsee vain käynnistää projekti sen sijaan, että koko lähdekoodi pitäisi kääntää uudestaan. (Buckland 2005)

Skriptaus mahdollistaa muutakin kuin pelkästään muuttuja-arvojen muuntelemisen. Mitä kehittyneempää skriptauskieltä käytetään, sitä monimutkaisempia kanssakäymisiä skriptauksella voi tehdä pelimoottorin lähdekoodin kanssa. Tämä tarkoittaa, että skriptit voivat sisältää esimerkiksi pelilogiikkaa, kuten tekoälyyn liittyviä asioita. Tämän lisäksi näiden avulla voi luoda uutta sisältöä peliin. (Buckland 2005)

Skriptaus on yleisesti ottaen tekstipohjaista. Muutamia yleisesti käytettyjä skriptikieliä ovat esimerkiksi Lua ja XML. Vaihtoehtoisesti pelinkehittäjät voivat myös luoda oman skriptikielensä. Tämä mahdollistaa sen, että pelinkehittäjät voivat muokata skriptauksesta juuri tarpeidensa mukaista. Oman kielen toteuttaminen vie kuitenkin paljon enemmän kehitysaikaa

kuin valmiin kielen käyttäminen. Yleisesti ottaen valmiiden kielten etu on näiden optimaalinen toteutus ja lähes virheetön tila pitkän kehitysajan johdosta. (Millington 2006)

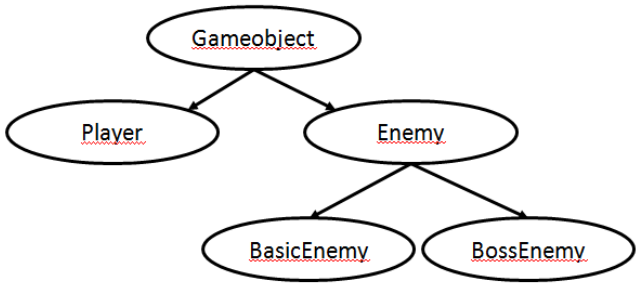
Skriptit ovat yleisesti ottaen joko ”tulkattuja” tai ”käännettyjä”. Tulkkaaminen tarkoittaa sitä, että skriptitiedosto on tekstimuotoinen ja yleisesti ottaen helposti luettava katsojan silmiin. Tämän takia pelimoottorissa pitää olla eräänlainen tulkkaaja, joka muuntaa kyseiset tiedostot pelimoottorin ymmärtämään muotoon rivi kerrallaan. Kääntäminen puolestaan tarkoittaa sitä, että skriptikielen kääntäjä kääntää valmiiksi skriptitiedoston konekieliseen muotoon. Tätä käännettyä tiedostoa voi pelimoottori suoraan hyödyntää ilman sen suurempia tulkkauksia. (Buckland 2005)

Skriptit mahdollistavat nopean peliarvojen muokkauksen ja muutkin kuin ohjelmoijat voivat täten muokata peliin liittyviä asioita. Tämän lisäksi nämä mahdollistavat myös nopean skaalaamisen pelin sisällölle. Tästä syystä skriptien käyttäminen voi säästää aikaa ja lisätä tuotteliaisuutta, mutta asialla on myös omat varjopuolensakin. Skriptien lukeminen on hitaampaa kuin suoraan lähdekoodista koodin tulkitseminen. Sen lisäksi jokaiselle skriptille täytyy myös tehdä omat vastineensa pelimoottoriin, jotta peli osaa hyödyntää tietoa oikein. Hitaus voi olla ongelma toteutustavasta riippuen, mutta nykyaikaisissa isommissa projekteissa skriptien käyttäminen on melkolailla välttämätöntä. (Buckland 2005)

6.2 Komponentti- ja oliopohjainen entiteetti

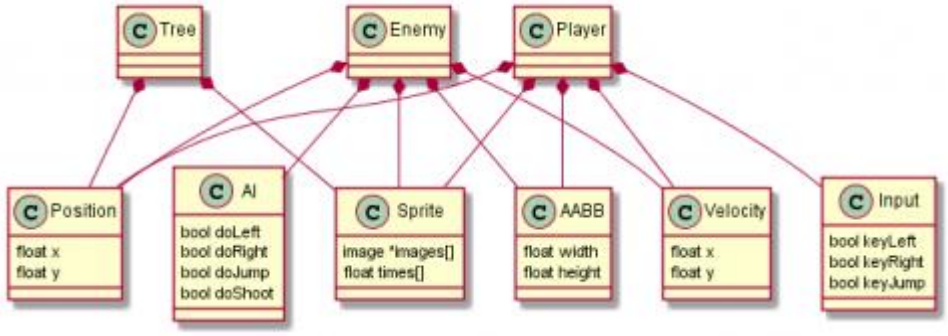
Komponentti- ja oliopohjaiset toteutustavat kuvastavat pelimoottorin tapaa toteuttaa pelientiteettejä, kuten hahmoja, vihollisia tai muita pelissä esiintyviä asioita. Tässä asiassa pitää ottaa huomioon, että pelimoottori tukee pääasiassa vain jompaakumpaa vaihtoehtoa.

Oliopohjainen toteutus on perinteisempi tapa toteuttaa entiteettejä. Oliopohjaisuudelle ominainen piirre on luokkarakenteiden perinnöllisyys (Kuvio 9). Jokainen entiteetti on eräänlainen olio, joka pystyy saamaan ominaisuuksia muilta entiteeteiltä perinnän avulla. Pidemmän päälle tämä tapa saattaa johtaa erittäin sekaviin luokkarakenteisiin. Siinä missä entiteettien määrä kasvaa, niin sitä vaikeampaa on sijoittaa uusia entiteettejä olemassa olevien sekaan. Varsinkin jos entiteetti tarvitsee useanlaista erilaista toiminnallisuutta. (Boreal Games 2013)



Kuvio 9. Yksinkertainen oliopohjainen luokkakaavio

Komponenttipohjaisuus on uudempi toteutustapa ja siinä entiteettejä rakennetaan eräänlaisen ”kokoamisen” – periaatteen mukaan perinteisemmän perimisen sijaan. Pohjimmiltaan tämä tarkoittaa sitä, että entiteetit ovat enemmänkin yhdistelmä erilaisia komponentteja, eli tietorakenteita (Kuvio 10). (Boreal Games 2013)



Kuvio 10. Yksinkertainen komponenttipohjainen luokkakaavio (Boreal Games 2013)

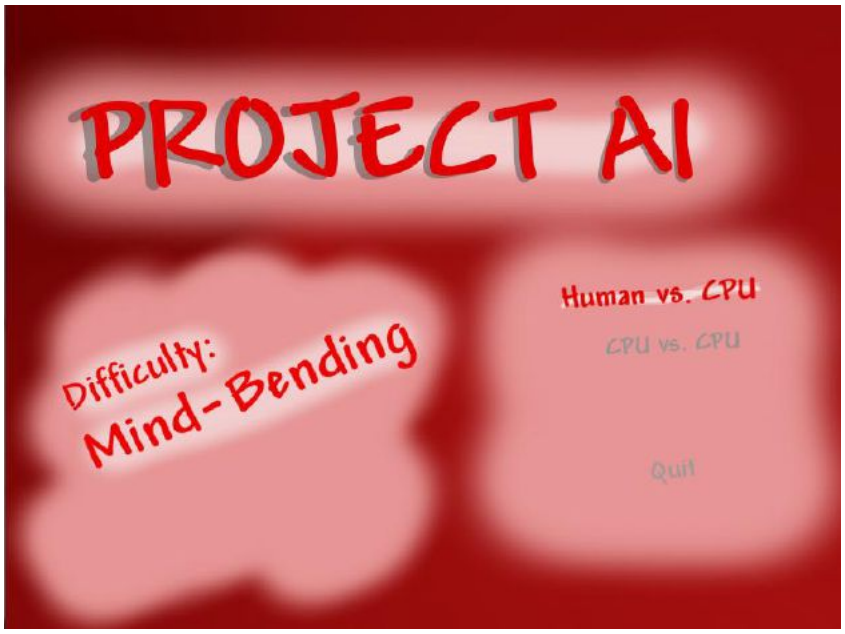
Yksittäinen komponentti sisältää vain tietoa sen sijaan, että siinä itsessään olisi pelilogiikkaa. Yksinään komponentit ovat melkolailla hyödyttömiä, joten näiden todellinen hyöty tulee esille, kun niihin yhdistetään komponentteja hyödyntävät järjestelmät. Yksi järjestelmä voisi olla esimerkiksi liikkuminen, joka hyödyntää sijainti- ja kiihtyvyyskomponenttia. Jos entiteetiltä puuttuu jompikumpi komponenteista, niin tällöin se ei käytä kyseistä järjestelmää. Jos kummatkin puolestaan löytyvät, niin tällöin entiteetti kykenee liikkumaan. (Boreal Games 2013)

Komponenttipohjaisessa toteutustavassa skaalattavuus, ominaisuuksien lisäys ja hallinta ovat parempia verrattuna oliopohjaiseen toteutukseen. Tästä syystä komponenttipohjaisuus on kasvattanut suosiotaan varsinkin isommissa ja monimutkaisemmissa peliprojekteissa. Kumpikaan toteutustapa ei ole kuitenkaan väärä, vaan kummallekin on omat tarkoituksensa. Valinta näiden välillä riippuu täysin pelinkehittäjien mieltymyksistä ja tarpeista. (Boreal Games 2013)

7 PROJECT AI:N ESITTELY

Project AI on tätä opinnäytetyötä varten luotu peliprojekti. Project AI on kaksin pelattava ”Shoot ’em Up” (shmup) - peli, jossa pelaajat pelaavat toisiaan vastaan. Peliprojektin inspiraationa on peli nimeltä ”Touhou 9: Phantasmagoria of Flower View”. Tästä syystä Project AI:n pelimekaniikat ovat käytännössä yksinkertaistetut versiot Touhou 9:n pelimekaniikoista. Projektilla ei ole kaupallista tarkoitusta, joten graafinen ulkoasu on hyvin yksinkertainen ja pelkistetty. Peliprojekti itsessään on toteutettu C# - ohjelmointikieltä ja XNA - kehitysympäristöä käyttäen sekä olio-ohjelmoinnin periaatteita noudattaen. Pelin grafiikat on toteutettu GIMP – ilmaiskuvankäsittelyohjelmaa käyttäen.

Project AI:ssa on kaksi eri pelimuotoa (Kuvio 11), jotka ovat pelaaja vastaan tietokone ja tietokone vastaan tietokone. Tämän projektin pääasiallisena tarkoituksena on, että pelaaja pelaa teoriataustan pohjalta kehitettyä tekoälyä vastaan. Vaihtoehtoisesti pelaaja voi myös katsoa tietokonetekoälyjen kamppailua toisiaan vastaan toisessa pelimuodossa. Pääideana kuitenkin on, että pelissä esiintyy tietokonetekoäly, joka pystyy sopeutumaan muuttuviin olosuhteisiin. Tärkeänä osana tätä projektia on, että tekoälyn luomiseen on hyödynnetty teoriaosuuden tietoja.



Kuvio 11. Pelin alkuvalikko

7.1 Pelin eteneminen

Kun pelaaja aloittaa pelin valitsemalla jommankumman pelimuodoista, niin hänen eteensä ilmestyy peliruutu (Kuvio 12) jossa on kaksi pelikenttää. Pelialueiden lisäksi ruudulla näkyy myös ruudunpäivitysnopeus ja peliaikaa kuvastava ajastin. Kummallakin pelaajalla on oma pelialueensa, jossa pelaajan tarkoituksena on yrittää pysyä hengissä. Pelaajat voivat vaikuttaa toistensa pelialueiden tapahtumiin hyödyntämällä useita eri pelimekaniikkoja, joista kerrotaan lisää seuraavassa osiossa.

Peli voi päättyä ainoastaan silloin kun toiselta pelaajalta loppuvat kestopisteet. Kun peli päättyy, peliruutuun ilmestyy lopetusruutu (Kuvio 13), jossa kerrotaan voittaja. Nappia painamalla pelaaja voi joko pelata uuden pelin tai palata takaisin alkuvalikkoon.

Kuvio 12. Peliruutu

Kuvio 13. Pelin lopetusruutu

7.2 Pelimekaniikat

Peli sisältää paljon erilaisia pelimekaniikkoja, joita pelaaja voi hyödyntää voiton tavoittelemiseksi. Pääasiassa nämä voi jakaa kolmeen eri kategoriaan: pelialueeseen, pelaajahahmoon ja vihollishahmoihin. Jokainen pelimekaniikka liittyy jollakin tavalla yhteen tai useampaan näistä kolmesta kategoriasta.

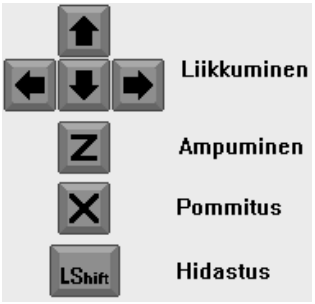
Kummallakin pelaajalla on oma pelialueensa (Kuvio 12). Pelialue sisältää pelaajan, erityyppisiä vihollisia ja ammuksia, joita pelaajan pitää väistellä. Pelialueen tapahtumiin vaikuttaa se, kuinka pelaaja itse käyttäytyy ja kuinka vastustajapelaaja toimii omalla alueellaan. Jotta näitä kanssakäymisiä voidaan kuvailla, niin pitää selventää pelaajahahmon ja vihollishahmojen ominaisuuksia tarkemmin.

7.3 Pelaajahahmo

Pelaajahahmon (Kuvio 14) oleellisimmat ominaisuudet ovat tämän kyky liikkua, ampua ja pommittaa. Jokaiselle pelaajan toiminnolle on vastaavat näppäimensä (Kuvio 15). Pelaaja kykenee liikkumaan kahdeksaan eri suuntaan ja tämä liikkuu aina samalla vauhdilla ilman kiihtyvyyttä. Eli kun liikkumisnappia painaa, niin pelaaja liikkuu suoraan maksiminopeudella ilman vauhdin erillistä nopeutumista tai hidastumista.

Pelaaja kykenee kuitenkin hidastamaan liikkumisnopeuttaan painamalla hidastamisnappia samanaikaisesti. Tämä mahdollistaa pelaajalle hienovaraisemman ja tarkemman liikkumisen kun siihen on tarvetta. Käytännössä hidastaminen puolittaa liikkumisnopeuden. Samaan aikaan kun liike on hitaampaa, niin pelaajahahmon päälle ilmestyy pelaajan osuma-aluetta kuvastava grafiikka (Kuvio 14). Tämä alue on ainoastaan näkyvillä silloin kun pelaaja painaa hidastusnäppäintä.

Kuvio 14. Pelaajahahmo ilman osuma-aluetta ja tämän kanssa



Kuvio 15. Pelaajan ohjausnäppäimet

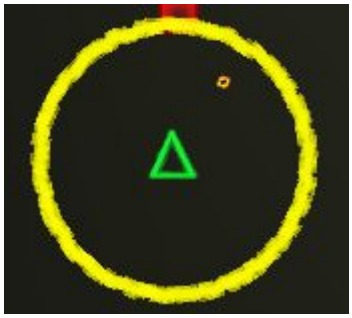
Osuma-alue kuvastaa sitä aluetta josta pelaaja ottaa vahinkoa. Käytännössä tämä tarkoittaa sitä, että jos pelialueen ammus osuu pelaajaan, niin tällöin tämä menettää yhden kestopisteen. Kun pelaaja ottaa vahinkoa, niin tämä on immuuni vahingolle hetken aikaa. Tätä niin sanottua kuolemattomuusaikaa kuvastaa pelaajahahmon välkkyminen. Pelaajalla on kolme kestopistettä, joita kuvastava grafiikka on pelialueen ylälaidassa (Kuvio 12, Kuvio 16). Jos pelaajan kestopisteet menevät nollaan, niin tällöin kyseinen pelaaja on hävinnyt pelin.

Kuvio 16. Kestopistegrafiikka

Jotta pelaaja kykenee vaikuttamaan pelialueen tapahtumiin, niin tällä on kyky ampua (Kuvio 17) painamalla ampumisnäppäintä. Ampumisnäppäintä voi pitää pohjassa, jolloin pelaaja ampuu koko ajan tietyin väliajoin. Vihollisia tuhoamalla pelaaja voi vaikuttaa vastustajan pelialueeseen, mutta tästä tarkemmin vihollisosiossa. Pelaajan ammuksille ei ole erillistä resurssia tai rajoitetta, joten pelaaja voi ampua niin paljon kuin haluaa.

Kuvio 17. Pelaajan ammukset

Viimeinen pelaajan ominaisuus ampumisen lisäksi on kyky pommittaa. Pommitus tarkoittaa sitä, että pelaaja voi hätätilanteessa käyttää pommia selviytyäkseen. Kun pommitusnäppäintä painaa, niin tällöin pelaajasta lähtee keltainen energia-aalto (Kuvio 18), joka tuhoaa kaiken tieltään. Pelaaja ei kuitenkaan voi pommittaa jatkuvasti, sillä tämä toiminto vaatii tarpeeksi ladattua voimaa pommitusmittarissa (Kuvio 19).



Kuvio 18. Pommitusominaisuus

Kuvio 19. Pommitusmittari ja ilmoitus

Pommitusmittaria kuvastaa grafiikka ruudun alalaidassa (Kuvio 12, Kuvio 19). Pelaaja voi käyttää pommia kun mittarista on vähintään puolet täynnä. Mittari täyttyy pikku hiljaa tuhoamalla vihollisia ja kun pommi on toimintavalmis, niin pelaajalle tulee ilmoitus asiasta (Kuvio 19). Pommilla on kaksi eri tasoa, jotka vaikuttavat pommin toimintaan. Ykköstason pommia voi käyttää kun mittari on puoliksi täynnä ja kakkostason pommia kun mittari on kokonaan täynnä.

Pommin taso vaikuttaa esimerkiksi siihen, kuinka laajalle alueelle tämä leviää. Tämän lisäksi jos pelaaja käyttää kakkostason pommin, niin tämä luo erikoisammuskuvion, joka lisää ammuksia vastustajan pelialueelle. Mittari kuluu käytetystä pommitasosta riippuen. Ykköstason pommi kuluttaa puolet mittarista ja kakkostason pommi kuluttaa koko mittarin.

7.4 Vihollishahmot

Pelialueella esiintyviä vihollisia on kolmea erilaista tyyppiä; perusvihollisia, erikoisvihollisia ja pomovihollisia. Jokaisella näillä on omat toimintonsa ja ehtonsa pelialueelle ilmestymiseen.

7.4.1 Perusviholliset

Yleisin pelialueella esiintyvä vihollistyyppi on perusvihollinen (Kuvio 20). Perusviholliset eivät itsessään tee mitään muuta kuin liiku paikasta A paikkaan B. Eli nämä eivät ammu pelaajaa kohti. Sen sijaan kun perusvihollinen kuolee pelaajan toimesta, niin tämä toiminto luo ammuksia (Kuvio 21) vastustajapelaajan pelialueelle.

Kuvio 20. Perusviholliset

Kuvio 21. Perusammukset

Pienemmät perusviholliset luovat pienempiä ammuksia ja isompi vihollinen luo isomman ammuksen. Näitä luotuja ammuksia on kahta eri tyyppiä (Kuvio 21). Violetit ammukset saavat suunnakseen sen hetkisen pelaajan sijainnin. Siniset ammukset sen sijaan saavat satunnaisen suunnan. Kun ammus luodaan, niin tälle arvotaan ammustyyppi ja tämä sijoitetaan satunnaiseen sijaintiin ruudun ylälaitaan. Lisäksi ammuksien liikkumisnopeus vaihtelee hieman, jotta tämä lisäisi satunnaisuutta näiden toimintaan.

Perusviholliset ilmestyvät pelialueelle aina ryhmässä, jolla on sama reitti. Pelissä on kymmeniä eri reittivaihtoehtoja, joista arvotaan yksi. Tähän ryhmään kuuluu neljä pienempää ja yksi isompi perusvihollinen. Perusvihollisryhmän ilmestymiselle on muutama ehto. Ryhmä ilmestyy joko silloin, kun pelialue on tyhjillään vihollisista tai kun tietty aika on kulunut edellisen ryhmän ilmestymisestä. Kummallekin pelaajalle ilmestyy samat vihollisryhmät samoilla reiteillä samassa järjestyksessä. Tämä tarkoittaa sitä, että vaikka toinen pelaaja päihittää vastustajia nopeammin, niin kummallekin pelaajalle luodaan silti samat vihollisryhmät, tosin vain eri aikoihin.

Tämän mahdollistamiseksi pelissä käytetään jonototeutusta perusvihollisten luontiin. Kun toinen pelaaja täyttää jommankumman ehdoista vihollisryhmän luontiin, niin kummankin pelialueen jonon jatkoksi luodaan tarvittavat tiedot seuraavasta vihollisryhmästä. Kun pelaajalla on tarvetta saada uusi vihollisryhmä alueelle, niin jonosta poimitaan ensimmäinen ja tämän tietojen pohjalta luodaan uusi vihollisryhmä. Pelialueiden vihollisryhmissä on kuitenkin hieman satunnaisuutta mukana, sillä niiden liikkumisnopeus vaihtelee hieman.

7.4.2 Erikoisviholliset

Perusvihollisten lisäksi on erikoisvihollisia (Kuvio 22), joilla on täysin oma toimintaperiaate. Erikoisvihollisetkaan eivät ammu pelaajaa kohti, mutta näiden ammuksien luontijärjestelmä eroaa paljonkin perusvihollisista. Erikoisvihollisia on kolmea eri tyyppiä; pieni, keskikokoinen ja iso.

Erikoisvihollisia ilmestyy pelialueelle kun pelaaja tuhoaa ison perusvihollisen. Tällöin pelaajan omalle alueelle ilmestyy pieni erikoisvihollinen. Erikoisviholliset ilmestyvät aina pelialueen ylälaitaan ja saavat hiukan satunnaisen suunnan alaspäin. Kun pelaaja tuhoaa pienen erikoisvihollisen, niin tämä toiminto luo vastustajan pelialueelle keskikokoisen erikoisvihollisen. Jos vastustaja tuhoaa tämän keskikokoisen vihollisen, niin tämä puolestaan luo ison erikoisvihollisen takaisin toisen pelaajan pelialueelle. Kun tämä iso erikoisvihollinen lopulta tuhotaan, niin tämä toiminto luo vastustajan pelialueelle isomman erikoisammuskuvion (Kuvio 23).

Kuvio 22. Erikoisviholliset

Kuvio 23. Erikoisammuskuvio

Koska isomman erikoisvihollisen saanti omalle pelialueelle on hankalaa, niin tästä syystä tämän tuhoaminen on palkitsevaa. Tätä mekaniikkaa voi pelaaja esimerkiksi hyödyntää siten, että jättää tuhoamatta keskikokoisia erikoisvihollisia, jolloin vastustaja ei pääse käsiksi erikoisammuskuvioihin. Tämä saattaa kuitenkin olla haastavaa kaoottisessa pelialueessa.

7.4.3 Pomovihollinen

Viimeisenä vihollistyyppinä on harvinaisempi pomovihollinen (Kuvio 24). Pomovihollinen ilmestyy tietyin väliajoin pelialueelle, jolloin pelaaja voi yrittää tuhota tämän. Pelaaja saa ilmoituksen (Kuvio 25) vähän ennen kuin pomovihollinen ilmestyy pelialueelle. Pomovihollisen tarkoituksena on luoda pelaajalle lisää haastetta ja täten luoda lisää taktikointimahdollisuuksia pelaajille.

Kuvio 24. Pomovihollinen

Kuvio 25. Ilmoitus pomovihollisesta

Pomovihollinen ilmestyy ruudun ylälaitaan, jonka jälkeen tämä alkaa laskeutumaan hiukan alaspäin pelaajaa kohti. Pomovihollinen on immuuni vahingolle laskeutumisen ajan. Kun pomo on pysähtynyt, niin tämä alkaa ampua omia ammuksiaan pelikentälle pelaajan riesaksi. Pomolla on omanlainen ammuskuviojärjestelmä, jota se noudattaa. Vähän ajan kuluttua ampumisen aloituksesta, pomovihollinen alkaa poistua pikkuhiljaa takaisin ruudun ylälaitaan.

Jos pelaaja onnistuu tuhoamaan pomovastuksen, niin tämä luo pelialueelle kerättävän esineen (Kuvio 26). Tämä kerättävä esine laskeutuu vähitellen pelialueen alalaitaa kohti. Jos pelaaja onnistuu keräämään tämän esineen, niin tällöin pelaajan pommitusmittari täyttyy maksimiin. On tietenkin mahdollista, että pelaaja ei ehdikään tuhota pomoa, jolloin kerättävä esine jää saamatta. Pitää ottaa myös huomioon, että jos pelaaja tuhoaa pomon käyttäen pommia, niin tällöinkään kerättävää esinettä ei luoda pelialueelle.

Kuvio 26. Pomovihollisen luoma esine

8 PELAAJATEKOÄLYN SUUNNITELMA

Ennen kuin kuvaillaan tarkemmin Project AI:n pelaajatekoälyn suunnitelmaa, on hyvä ensin tutustua shmup-peligenren tekoälyyn yleisellä tasolla.

8.1 Tekoäly Shmup -peleissä

Pelaajatekoälyn esiintyminen shmup -peleissä on erittäin harvinaista. Näitä on pääasiassa vain kilpailullisemmissa shmup -peleissä, joissa ideana on kamppailla toisen pelaajan kanssa. Tällaisia pelejä on esimerkiksi ”Twinkle Star Sprites” (Kuvio 27) ja ”Touhou 9: Phantasmagoria of Flower View” (Kuvio 28). Perinteisemmissä shmup -peleissä on joskus mahdollisuus kahdenpelaajan tiimipelaamiseen. Tällöin on yleistä, että toista pelaajaa pystyy ohjastamaan vain ihminen, tietokonetekoälyn sijaan.

Kuvio 27. Twinkle Star Sprites

Kuvio 28. Touhou 9.

Pelaajatekoälyn sijaan perinteisemmissä shmup-peleissä on pääpaino vihollistekoälyn puolella. Vihollistekoäly voidaan näissä yleisesti ottaen luokitella perusvihollisiin ja pomovihollisiin, sekä näiden luomiin ammuskuvioihin. Shmup-tekoäly on pääasiassa hyvin pitkälti ennalta määriteltyä ja yksinkertaista tekniseltä kannalta.

Perusviholliset käyttäytyvät aina samalla tavalla, liikkuen paikasta A paikkaan B ja samalla ampuen omaa ammuskuviotaan. Näihin voidaan kuitenkin luoda hieman satunnaisuutta lisäämällä näille vaihtoehtoisia käyttäytymismalleja riippuen siitä, missä pelaaja sijaitsee pelialueella sillä hetkellä. Shmup-peleissä keskitytään yleisesti ottaen erittäin paljon pomovihollisten

tekoälyyn. Pomoviholliset koostuvat useista eri vaiheista, joissa kussakin pomovihollinen luo uniikkeja ja monimutkaisia ammuskuvioita pelaajan päihittämiseksi. Shmup-pelien pääasiallinen haaste tuleekin siitä, että pelaajan pitää keksiä kuinka ammuskuvioiden ammukset pystyy väistämään.

Koska shmup-peleissä vihollistekoäly itsessään on mekaanisesti yksinkertaista, niin peleissä on useita eri vaikeusasteita luomassa lisää haastetta pelaajalle. Vaikeusaste lisää yleisesti ottaen perusvihollisten määrää kentissä ja nämä ampuvat joko enemmän tai monimutkaisempia ammuskuvioita. Pomovastuksetkin vaikeutuvat entistä monimutkaisempien ammuskuvioiden myötä. Näitä pomoviholliston ammuskuvioita saattaa olla myös enemmän vaikeusasteesta riippuen.

8.2 Project AI:n pelaajatekoäly

Pelaajatekoälyn tekemisessä shmup-peliin on omat haasteensa. Pelitekoälyn suunnittelemisessa yleisestikin pitää ottaa paljon asioita huomioon, mutta shmup-peleissä pelimekaniikat tuovat lisää päänvaivaa. Shmup-pelaajatekoälyn saa hyvin helposti näyttämään siltä, että tämä huijaa tai tämän käyttäytyminen näyttää luonnottomalta.

Project AI:n pelaajatekoälyn pääasiallisena tarkoituksena on tarjota sopivan haastava vastus pelaajalle. Tämä tarkoittaa sitä, että tekoäly ei ole liian helppo, mutta ei myöskään mahdoton päihittää. Tekoälyn toteutuksen kannalta ensisijainen tavoite on kuitenkin, että se toimii asiallisesti sen sijaan että saman tien yritetään tehdä siitä ideaalinen. Tämä tarkoittaa sitä, että tekoäly osaa käyttäytyä mekaanisesti kuten sen on ajateltu. Kun tekoäly toimii alustavasti oikein, niin tämän jälkeen siitä voi tehdä joko haastavamman tai inhimillisemmän. Tämä inhimillisyys voi olla esimerkiksi sellaista, että päätöksenteossa on omat virhemarginaalit, jolloin tekoäly saattaa ottaa turhia riskejä tai arvioida ammuksien lentoratoja väärin.

Inhimillisyyden toteuttamisessa on kuitenkin omat haasteensa, johtuen tekoälyn mekaanisesta käyttäytymisestä. Jos tekoäly käyttäytyy normaalisti lähes täydellisesti, niin nämä yllättävät tyhmät virheet saavat tämän käyttäytymisen näyttämään typerältä. Tästä johtuen tähän peliprojektiin voi lisätä muutamia vaikeusasteita. Tällöin pelaajatekoäly käyttäytyy näiden vaikeusasteiden mukaisesti, jolloin tekoäly voi olla erittäin haastava tai sellainen joka tekee virheitä

vaikeusasteesta riippuen. Toisaalta vaikeusasteille ei välttämättä ole tarvetta, riippuen siitä miten tekoälyn toteutusprosessi etenee.

8.2.1 Käyttäytymissuunnitelma

Project AI:n pelaajatekoälyllä tulee olemaan useita eri toimintatavoitteita. Tekoälyllä voi olla yksi tavoite kerrallaan, mutta jos olosuhteet vaihtuvat, niin tämä kykenee muuttamaan suunnitelmiaan lennosta.

Tärkein toiminnallinen tavoite käyttäytymisen kannalta on kuitenkin se, tekoäly kykenee väistelemään ammuksia. Jotta väistely olisi mahdollista, tekoälyn pitää analysoida pelialueen muutoksia sekä laskea ja ennakoida ammuksien lentoratoja. Analysoinnin ohella oleellisena osana tekoälyä on sen reitinhakukyky. Kaikki tekoälyn toimintatavoitteet hyödyntävät kentän analysointia ja reitinhakua päätöksenteossaan.

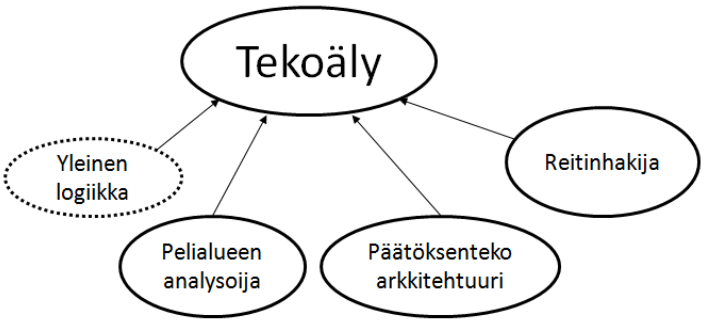
Ammuksien väistelyn lisäksi tekoälyn pitää yrittää tuhota vihollisia. Jos vihollisten jahtaaminen on turvallista, niin tekoäly suunnittelee reittejä, jotka mahdollistavat vihollisten tuhoamisen. Eri vihollistyypeillä on erilaiset prioriteetit tuhoamiselle. Isoin prioriteetti on pomovihollisella, sillä tämän tuhoaminen on palkitsevinta. Pomovihollisen jälkeen ovat erikoisvihollliset ja prioriteettilistan pohjimmaisena ovat perusvihollliset.

Viimeisenä toimintatavoitteena on esineiden keruu. Jos pelialueelle ilmestyy kerättävä esine, niin tekoäly pyrkii aina keräämään sen. Tekoäly voi kuitenkin hylätä tämän tavoitteen, jos esineen kerääminen osoittautuu liian vaaralliseksi.

Tärkein piirre tekoälyn kannalta on se, että sillä on samat ominaisuudet kuin ihmispelaajallakin. Tämä tarkoittaa sitä, että tekoäly ja ihmispelaaja liikkuvat, ampuvat ja pommittavat samalla tavalla. Tämä on tärkeää siinä mielessä, että tekoäly ei kykene huijaamaan tai saamaan etua ihmispelaajaan nähden pelimekaniikkojen avulla.

8.2.2 Tekninen suunnitelma

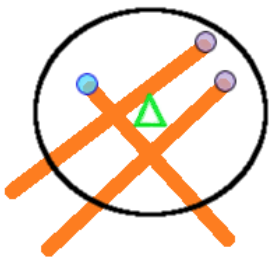
Jotta käyttäytymissuunnitelman toteutus on mahdollista, niin pelitekoälyn logiikka koostuu kolmesta eri osasta (Kuvio 29). Nämä osat ovat pelikentän analysaattori, päätöksentekoarkkitehtuuri ja reitinhakumetodi. Jokaiselle näistä on useita eri toteutustapoja, joten oikea tapa löytyy vain testaamalla suunniteltuja vaihtoehtoja.



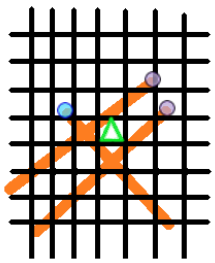
Kuvio 29. Yleisen tason kaavio tekoälyn rakenteesta

Pelikentän analysointi on yksi oleellisimmista tekoälyn komponenteista. Tekoälyn päätöksenteko riippuu hyvin pitkälti siitä, kuinka tekoäly analysoi pelialuetta. Pelikentän analysoimiseen on muutama eri toteutustapa. Ensimmäinen vaihtoehto on pelaajan ympärillä oleva kehä (Kuvio 30), jonka sisältävää aluetta analysoidaan. Tämä tarkoittaa sitä, että tekoäly ei analysoi koko pelialuetta, vaan vain pientä osaa siitä. Tähän voi toki lisätä komponentteja, jotka antavat tekoälylle tietoa esimerkiksi kentälle ilmestyvistä esineistä tai muita oleellisia tietoja.

Toinen tapa toteuttaa pelikentän analysointi on jakaa koko pelialue ruudukoksi (Kuvio 31). Tällöin tekoäly on koko ajan tietoinen siitä, että mitä pelialueella tapahtuu. Tämä lähestymistapa on periaatteessa järkevämpi toteuttaa, sillä tätä ruudukkoa voi hyödyntää suoraan muissakin tekoälyn komponenteissa. Esimerkiksi reitinhaku on yksinkertaista toteuttaa jos tämän apuna on valmis ruudukko, johon on merkattu tarvittavat tiedot.



Kuvio 30. Kehä analysointi konsepti

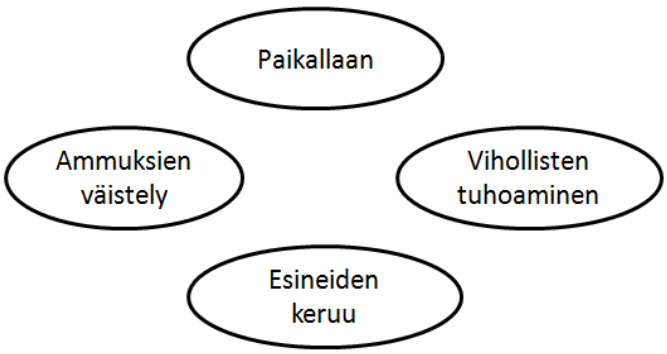


Kuvio 31. Ruudukko analysointi konsepti

Pelikentän analysoinnin toteutuksessa pitää kuitenkin ottaa huomioon, että se on erittäin raskasta prosessointitehon kannalta. Analysoitavia asioita voi olla erittäin paljon, riippuen pelialueen tapahtumista ja kaoottisuudesta. Tästä syystä analysaattoria toteutettaessa pitää miettiä, kuinka siitä saa mahdollisimman kevyen suorituskyvyn kannalta.

Pelikentän analysointia hyödyntää tekoälyn päätöksentekoarkkitehtuuri. Tämän arkkitehtuurin voisi toteuttaa joko yksinkertaisella tilakoneella tai käytöspuurakenteella. Näihin kahteen voi myös yhdistää toimintojen hyötyarvojen laskentaa, jotta päätöksenteosta saa hiukan moninaisemman.

Käyttäytymissuunnitelman perusteella tekoälyn eri tiloja (Kuvio 32) on kuitenkin loppujen lopuksi melko vähän. Tästä syystä arkkitehtuuri tulee olemaan joko yksinkertainen tilakone, hyötypohjainen laskenta tai näiden kahden yhdistelmä. Lopullisen menetelmän löytämiseksi pitää testata näiden toteutusta käytännössä. Jokaisella tekoälytilalla on omat toimintonsa ja ehdot näiden aktivoitumiseen. Pääideana kuitenkin on, että tekoäly voi olla vain yhdessä tilassa kerrallaan ja mistä tahansa tilasta voi siirtyä mihin tahansa muuhun tilaan.



Kuvio 32. Tekoälyn tilat

Viimeisenä tekoälyn komponenttina on tämän reitinhakukyky. Jokaisessa tekoälyn tilassa hyödynnetään tavalla tai toisella reitinhakukykyä. Esimerkiksi ”ammuksien väistely” -tilassa yritetään löytää mahdollisimman läheltä turvallisia kohtia ja reittejä kyseisiin pisteisiin. ”Vihollisten tuhoamisessa” puolestaan pyritään liikkumaan siten, että tekoäly on ampumalinjassa pelialueella oleviin vihollisiin nähden.

Reitinhakumetodi riippuu hyvin paljon siitä, kuinka pelialueen analysoimisen toteuttaa. Jos kenttä jaetaan ruudukoksi, niin reitinhakuun voi käyttää suoraan joko A* tai Dijkstra -metodia. Jos reitillä on selkeä päätepiste, niin A* on paras vaihtoehto. Tosin tämän tyylisessä pelissä myös Dijsktra-metodiakin voi hyödyntää erittäin paljon, jollei se osoittaudu liian raskaaksi. Esimerkiksi jos tekoälyn pitää löytää lähin turvallinen piste, niin Dijkstra-metodin avulla voi etsiä tämän kätevästi. Toisaalta jos tämä turvallinen piste on ennalta tiedossa, niin tällöin A* -metodin avulla voi laskea reitin suoraan tähän.

Ruudukkopohjainen liikkuminen saa tosin tekoälyn liikkumisen näyttämään hiukan kankealta. Tärkein asia reitinhaun ja minkä tahansa muunkin komponentin kannalta on kuitenkin sen toiminnallisuus. Sen jälkeen kun komponentti toimii oikein, siitä voi tehdä paremman tai hienomman.

9 PROJEKTIN TOTEUTUSPROSESSI

Kuten peliprojekteille on ominaista, projektin toteutus koostui useasta eri vaiheesta. Ennen varsinaisen pelin ja tekoälyn toteutuksen aloittamista piti suunnitella näihin liittyviä asioita ja mekaniikkoja ennakkoon. Tätä suunnitteluvaihetta kutsutaan esituotantovaiheeksi. Projektin suunnittelutyö ei kuitenkaan pääty tähän vaiheeseen, sillä suunnittelu ja pelimekaniikkojen muokkaaminen jatkuu koko projektin ajan.

Koska tämä peliprojekti sai inspiraationsa olemassa olevasta pelistä, niin tämän projektin pelimekaniikkoja ei tarvinnut suunnitella tyhjästä. Kyseisen pelin intensiiviseen pelaamiseen käytettiin muutama päivä. Tämän pelailun tarkoituksena oli tutkia pelimekaniikkoja ja ottaa selville miten nämä tarkalleen toimivat. Tämän jälkeen piti suunnitella, miten näitä mekaniikkoja yksinkertaistaisi tähän peliprojektiin.

Tämän suunnittelutyön jälkeen oli peliprojektin toteutuksen vuoro. Tällä projektilla ei ollut aikaisempaa pohjaa, joten projektin toteutus lähti ruohojuuritasolta liikkeelle. Tämän projektin toteuttamiseen käytettiin projektipohjaa, jota on kehitelty useamman aikaisemman projektin ajan. Tämä projektipohja nopeutti kehitystyötä, sillä siinä on useita projekteille ominaisia perusominaisuuksia valmiiksi toteutettuna.

Pelin toteutus eteni varsin jouhevasti. Mitään sen kummempia ongelmia ei tullut vastaan, vaan sen sijaan päänvaivaa aiheutti se, miten asiat toteutettaisiin järkevästi ohjelmoinnin näkökannalta. Pelin toteutus eteni pelimekaniikka pelimekaniikalta ja pitkin projektia joitakin mekaniikkoja muuteltiin eri suuntaan alkuperäisestä suunnitelmasta. Esimerkiksi alun perin pommi tuhosi suoraan kaiken pelialueelta. Tätä mekaniikkaa muutettiin projektin edetessä siten, että kaiken tuhoamisen sijaan tuhoalueesta tehtiin ympyränmuotoinen. Tämä alue laajenee ja tuhoaa kaiken tieltään. Tämän uuden pommin kantama ei myöskään yllä koko ruudun alueelle. Tämän lisäksi pommiin lisättiin vielä voimakkuustasoja, jotka vaikuttavat pommin toimintaan.

Loppujen lopuksi suurin osa referenssipelin pelimekaniikoista päätyi peliin jollakin tasolla, jotkut erittäin yksinkertaistettuna, jotkut puolestaan sellaisenaan. Projektin suunnittelun ja toteutuksen kannalta olennaisin asia oli se, että aluksi projektiin tehdään vain kaikki perusominaisuudet toimiviksi. Tähän päädyttiin sen takia, että aikaa ei käytettäsi turhaan kaikennäköiseen hienosäätöön, vaikka projektissa muutoin riittäisi vielä tekemistä.

9.1 Pelaajatekoälyn toteutus

Kun projektissa oli kaikki perusominaisuudet jollain tasolla, niin tämän jälkeen oli pelaajatekoälyn toteutuksen vuoro. Pelaajatekoälyn toteutus lähti liikkeelle myös suunnittelutyöstä. Tässä vaiheessa oli tiedossa pelin olemassa olevat perusmekaniikat, joten käyttäytymisen suunnitteleminen oli helppoa. Tekniseltä kannalta on olennaista, että teoriataustan tietoja hyödynnettäisiin mahdollisimman paljon toteutusvaiheessa.

Kun alustava käyttäytymis- ja tekninen suunnitelma olivat valmiita, oli toteutuksen vuoro. Pelaajan ominaisuudet kuten ampuminen ja pommittaminen tehtiin alustavasti yksinkertaisiksi. Tekoäly ampuu automaattisesti koko ajan, riippumatta siitä onko vihollisia edessä vai ei. Pommitus puolestaan toimii siten, että jos ammus tai vihollinen on tarpeeksi lähellä tekoälyä, niin tämä pommittaa. Pommituksessa on samat ehdot kuin ihmispelaajallakin. Tekoäly ei kykene pommittamaan, ellei voimaa ole tarpeeksi kerääntynyt.

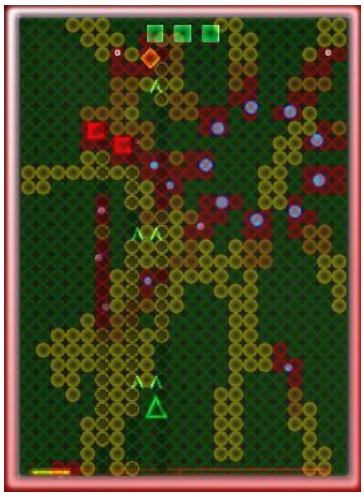
Muutoin tekoäly koostuu kolmesta isommasta komponentista, jotka ovat kentän analysoija, päätöksentekoarkkitehtuuri ja reitinhakumetodi. Toteutuksen kannalta oli olennaista, että komponenttien testaus ja mahdollinen virheenetsintä olisi mahdollisimman helppoa. Tästä syystä jokaiseen komponenttiin kehitettiin visuaaliset työkalut, jotka helpottivat kehitystyötä huomattavasti.

9.1.1 Pelialueen analysoija

Komponenttien toteutus lähti liikkeelle kentän analysoijasta. Tämä siitä syystä, että analysoija on periaatteessa tekoälyn tärkein ominaisuus, sillä muiden komponenttien toteutus riippuu hyvin paljon tästä komponentista. Alun perin suunniteltiin eräänlaista kehä- tai ruudukkopohjaista ratkaisua. Hyvin nopeasti tuli todettua, että ruudukkopohjaisuudella lähdetään liikkeelle, sillä tämä helpottaa muiden komponenttien toteutusta huomattavasti.

Loppujen lopuksi tämä analysoija toimii siten, että se jakaa koko pelialueen ruudukoksi (Kuvio 33). Yhdellä ruudulla on vaarallisuus- ja hyötyarvo, joita muutetaan pelialueen tapahtumien myötä. Pelin edetessä tähän analysoijaan syötetään tarpeelliset tiedot aina kun ammus, vihollinen tai esine luodaan pelialueelle. Ammuksien lentoratoja laskemalla tämä merkkaa

ruutuihin näiden vaarallisuusarvot. Vaarallisuusarvoja on neljä, jotka ovat vihreä, keltainen, oranssi ja punainen. Punainen on erittäin vaarallinen ja vihreä tarkoittaa turvallista ruutua.



Kuvio 33. Pelialueen analysoija

Vihollisista merkataan näiden sijainti vaaralliseksi ja tämän lisäksi vihollisen sijaintisarake merkataan hyödylliseksi. Eri vihollisilla on eri hyötyarvot, jotta päätöksenteossa ja reitinhaussa voidaan tehdä tarkempia valintoja. Samoin myös kerättävien esineiden sijaintisarake merkataan hyödylliseksi. Esineiden hyötyarvo on korkein pelissä.

Tämän analysaattorin toteutuksessa helpotti erittäin paljon visuaalinen testityökalu (Kuvio 33), sillä tämä havainnollistaa suoraan analysaattorin toiminnan. Varsinaisia ongelmia toteutuksen kannalta tuotti vain C# -ohjelmointikieli itsessään. C#:ssa ja C++:ssa muuttujat toimivat vähän eri tavalla. C#:ssa ei ole varsinaisia osoittimia, joten muuttujat toimivat välillä vähän miten sattuu kun niitä välitetään parametreina. Tästä sitten aiheutui kaikennäköistä päänvaivaa toteutuksen aikana, mutta kaikki ongelmat tuli ratkottua loppujen lopuksi.

Koska tämä kentän analysointi on varsin raskasta prosessointitehon kannalta, niin tämä komponentti toimii omassa säikeessään. Loppujen lopuksi tämä komponentti on varsin toimiva perustasoltaan, joten tämä luo hyvän pohjan kahdelle muulle komponentille. Vaihtoehtoisesti ruudukosta voi tehdä vähän erimuotoisen tai ruutuja voi pienentää, jos tästä haluaa tarkemman.

9.1.2 Päätöksentekoarkkitehtuuri ja Reitinhakumetodi

Päätöksenteko ja reitinhaku liittyvät toisiinsa hyvin paljon, joten näiden kehitys eteni rinnakkain. Jotta tehdyillä päätöksillä on merkitystä, niin tekoälyn tarvitsee pystyä liikkumaan asianmukaisesti.

Päätöksentekoon suunniteltiin alun perin joko tilakonetta tai hyötypohjaista laskentaa. Erinäisen testauksen tuloksena lopullinen toteutustapa päätyi tilakoneeksi. Hyötypohjaisen laskennan yhdistäminen tilakoneeseen oli yksi vaihtoehto, mutta tälle ei ollut tarvetta. Tilakoneella on neljä tilaa (Kuvio 34), kuten alun perin suunniteltiin. Tilakone voi olla yhdessä tilassa kerrallaan ja jokaiselle tilalle on omat siirtymäehtonsa. Koska tiloja vähän, niin jokaisesta tilasta voi siirtyä mihin tahansa muuhun tilaan.

Kuvio 34. Tilakoneen visualisointi

Kuvio 35. Reitinhaun visualisointi

Tilakoneen tekeminen oli varsin yksinkertaista, joten tämän toteuttamisessa ei ollut sen suurempia ongelmia. Visualisointi (Kuvio 34) näyttää kaikki tekoälyn tilat ja korostaa nykyistä tilaa. Toisaalta reitinhakumetodin toteuttaminen osoittautui näistä komponenteista ongelmallisimmaksi.

Reitinhaku toimii siten, että reitinhakijalle syötetään halutun ruudun tiedot sen hetkisestä tilasta riippuen. Visualisointi (Kuvio 35) piirtää tekoälyn sen hetkisen reitin valkoisina pisteinä. Jos tila näkee tarpeelliseksi, niin reitti luodaan uudestaan halutuilla parametreillä tilasta riippuen.

Reitinhaun toteutuksessa ilmeni usein ongelmia. Alun perin reitinhakua lähdettiin kokeilemaan Dijkstra -metodilla. Teoriatasolla tämä on varsin loogista, sillä tekoälyllä ei ole välttämättä tiedossa juuri sitä oikeaa ruutua, mihin tämä haluaa. Vaihtoehtoisesti ruutuja voi olla myös useita, joten jokaiseen pitäisi laskea erikseen reitit. Tällaiseen tarkoitukseen Dijkstra soveltuu hyvin. Kokeilun jälkeen huomattiin, että Dijsktra ei olekaan ihan niin hyvä vaihtoehto shmup-peliin, sillä se on vähän turhan raskas. Syynä saattoi olla hiukan virheellinen testitoteutustapa, mutta Dijkstra osoittautui tietyissä tilanteissa erittäin hitaaksi. Hitauden lisäksi se jäi myös jumiin ajoittain.

Dijskran jälkeen oli A* -metodin vuoro. Ongelmana kuitenkin on se, että A* toimii parhaimmillaan silloin, kun haluttu määränpää on ennalta tiedossa. Tästä syystä A* -metodin yhteyteen kehiteltiin lisäominaisuus auttamaan reitinhaussa. Eli ennen kuin A* metodilla etsitään reitti, niin reitinhakija etsii ruudukosta kaikki halutut ruudut ja lisää nämä listaan. Tämän jälkeen lista lajitellaan etäisyyden mukaan lähtöpisteestä. Tästä listasta poimitaan määränpääksi lähin ruutu ja tämän jälkeen A* metodilla lasketaan reitti tähän. Kun tätä ajattelee tarkemmin, niin tämä ratkaisu ei ole ihanteellisin vaihtoehto. Vaikka ruutu on teoriassa lähimpänä aloituspistettä, niin välttämättä haluttu reitti ei ole siltikään lyhin mahdollinen. Tästä syystä reitinhaku keskeytyy jos matkanvarrella huomataan, että löytyykin halutuilla ominaisuuksilla oleva ruutu.

Tämä A* -metodin viritelmä osoittautui toimivan paljon paremmin kuin aikaisempi Dijkstra -metodi. Kehityksen aikana tosin tässäkin ilmeni kaikennäköisiä ongelmia. Tämä jumiutui ajoittain ja virheenpaikantaminen oli työlästä, koska jumiutuminen oli satunnaista. Kaikki virheet johtuivat kuitenkin lopulta pienistä epäloogisuuksista reitinhakualgoritmissa. Kun nämä epäloogisuudet korjattiin ja lisäiltiin muutamia uusia asioita, niin reitinhaku toimi erittäin hyvin. Tällöin tuli huomattua, että oleellisin osa tekoälyn älykkyyttä on sen reitinhakukyky.

Tekoälyn visuaalinen käyttäytyminen pelaajan silmiin näyttää tässä vaiheessa vähän luonnottomalta. Toisaalta tekoäly toimii mekaanisesti alustavan suunnitelman mukaisesti. Eli tekoäly tarjoaa vastuksen, mutta tätä ei ole kuitenkaan mahdotonta voittaa. Projektin pelimekaniikat ovat kuitenkin suhteellisen yksinkertaiset, joten taitava pelaaja ei voi hävitä peliä. Jos nykyistä tekoälyä haluaa muokata, niin tähän on useita eri tapoja.

Tekoälyn käyttäytymisestä saa halutunlaisen muokkaamalla tilakoneen ominaisuuksia. Siirtymäehdot tilojen välillä ovat helposti muokattavissa, joten käytöstä saa älykkäämmäksi haluttaessa. Nykyistä tekoälyä saa myös paljon älykkäämmäksi, jos tämän reitinhakua muuttaa parempaan suuntaan. Reitinhaku on yllättävän suuri osa tekoälyn älykkyyttä, sekä teknisesti että visuaalisesti. Tarkoituksena ei kuitenkaan ollut luoda täydellistä tekoälyä, joten suunniteltuun tarkoitukseen tämä toteutettu tekoäly on varsin onnistunut. Parannusmahdollisuuksia on erittäin paljon, jos tätä halutaan ryhtyä kehittämään eteenpäin.

9.2 Jatkokehitys

Tässä vaiheessa peli on perusmekaniikoiltaan valmis ja pelaajatekoäly toimii alustavan suunnitelman lailla. Joten miten projektia voi kehittää parempaan suuntaan? Kaikennäköisiä pelimekaanisia parannuskohteita on erittäin paljon. Parannusvaraa on sekä pelimekaniikkojen että tekoälyn puolella.

Pelimekaniikkoja voi parantaa usealla eri tavalla. Ensinnäkin kerättävien esineiden määrää voi nostaa. Tällä hetkellä pelissä on vain yksi esine, joten uusien esineiden lisääminen tuo paljon vaihtelua. Uusi esine voi olla esimerkiksi ammuksia luova esine, joka voisi luoda joko erikoisammuskuvioita tai joukon normaaleja ammuksia vastustajan alueelle.

Esineiden lisäämisen lisäksi ammuskuvioissa on paljon parannettavaa. Erityisesti pomovihollisen ammuskuviota voi parantaa huomattavasti. Tällä hetkellä pomovihollinen roiskii ammuksia typerästi, joten tähän voisi kehitellä paljon paremman ratkaisun. Myös erikoisvihollisista syntyviä ammuskuvioita voisi lisätä enemmän vaihtelun vuoksi.

Ammuskuvioiden lisäksi itse ammuksetkin voisivat käyttäytyä eri tavalla. Sen sijaan että ne kulkisivat suorassa linjassa, niiden kulkusuunta voisi kaareutua vähitellen. Myös ammuksien luontilogiikkaa pystyisi parantamaan. Tämä tarkoittaa sitä, että peliin voi lisätä eräänlaisen ”Combo” -järjestelmän. Käytännössä tämä toimisi siten, että vihollisten tuhoaminen lyhyen ajan sisällä kasvattaisi eräänlaista kerrointa. Tämä kerroin puolestaan vaikuttaa luotujen ammuksien määrään. Kerroin nollaantuu, jos vihollisia ei tuhoa tietyn ajan sisään.

Pommitusominaisuudessakin on hieman säätövaraa. Siihen voisi esimerkiksi lisätä uusia voimakkuustasoja. Näihin voimakkuustasoihin voi lisätä kaikennäköisiä uusia ominaisuuksia.

Esimerkiksi voimakkuustasoja voisi olla kolme tai neljä. Muutamat ensimmäiset tasot luovat uusia erikoisammuskuvioita ja viimeinen voi luoda erikoispomoviholisen vastustajan alueelle. Tämän erikoispomon tuhoamisesta voi palkita pelaajan esimerkiksi kerättävällä esineellä. Myös jonkinlaisen automaattisen pommitusmekanismin voi kehittää. Esimerkiksi kun tuhoaa vihollisia niin saa pisteitä. Kun pisteitä on tarpeeksi, niin pelaaja käyttää ilmaisen pommin, joka luo joko ammuksia tai erikoispomoviholisen vastustajan alueelle.

Pelimekaanisten muutoksien lisäksi myös pelaajatekoälyssä on paljon parannettavaa. Jokaiseen kolmeen tekoälyn komponenttiin voi tehdä parannuksia jollain tasolla. Ensinnäkin kentän analysaattoria saa jo sillä paremmaksi, että lisää ruudun vaarallisuustasoja muutaman lisää. Näiden uusien vaarallisuustasojen myötä päätöksentekoarkkitehtuurista ja reitinhausta saadaan paljon monimuotoisemmat. Päätöksenteossa voi tehdä paljon hienovaraisempia päätöksiä näiden avulla. Reitinhaku puolestaan voi suunnitella paremmin reitit.

Nykyistä reitinhakumetodia voi muutenkin vaihtaa tai kehittää paremmaksi, sillä tämä nykyinen metodi ei ole ihan järkevin mahdollinen. Kentän analysaattoria voisi kehitellä myös huomattavasti kevyemmäksi prosessointitehon kannalta. Ihanteellisin tilanne on että analysaattorista saisi niin kevyen, että tämän ei tarvitsisi olla omassa säikeessään. Myös vaikeustasojen lisääminen peliin on yksi vaihtoehto lisätä erilaisia tekoälyjä.

10 POHDINTA

Miten opinnäytetyöprojekti kokonaisuudessaan lopulta onnistui? Muutamia ongelmia ilmeni saman tien aiheen päättämisen jälkeen teoriaosuuteen liittyen. Teoriatutkimusta tehdessä tuli huomattua, että shmup-pelien pelaajatekoälystä on melko mahdotonta löytää asiallisia lähteitä. Aihetta kuitenkaan ei haluttu vaihtaa, joten teoriaosuudessa kuvailtavan pelitekoälyn rinnastaminen käytännönosuuteen osoittautui hankalaksi. Tämän seurauksena teoriaosuuden toteutuksessa piti tehdä hieman kompromisseja. Teoriaosuudessa käsitellään pelitekoälyä yleisestä näkökulmasta ja käytännönosuuden puolella kuvaillaan shmup-pelien tekoälyä. Muilta osin teoriaosuus on varsin aiheeseen liittyvä.

Käytännönosuuden tekeminen puolestaan onnistui erittäin hyvin loppujen lopuksi. Opinnäytetyön tarkoituksena oli suunnitella ja toteuttaa toimiva pelaajatekoäly ja tässä tavoitteessa onnistuttiin. Tiukka aikataulu aiheutti hieman huolta käytännönosuuden toteutuksen kannalta projektin alkuvaiheilla. Tekoälyn toteutuksen lisäksi tähän projektiin piti toteuttaa itse pelikin, joten aluksi piti varautua siihen, että työtä joudutaan rajaamaan sisällön suhteen. Loppujen lopulta kaikki asiat saatiin perusmekaniikoiltaan toteutettua, joten onneksi ei ollut tarvetta rajata aihetta.

Opinnäytetyön ideana oli tutkia teoriaosuuteen useita eri tapoja toteuttaa tekoälyjä. Näitä tietoja hyödyntäen ja testaten piti toteuttaa oma tekoäly. Shmup-peligenre valittiin sillä perustein, että pelaajatekoäly on näissä harvinaista. Lisäksi ideana oli luoda tekoäly, joka pystyy sopeutumaan jatkuvasti muuttuviin tilanteisiin, joten shmup-peligenre sopii tähän tarkoitukseen erittäin hyvin.

Kuinka valittu toteutustapa lopulta soveltui tähän projektiin? Kuten teoriaosuudessa tuli tutkittua, niin tekoälyjen toteutukseen ei ole yhtä oikeaa tapaa. Sen sijaan on tärkeää, että pelinkehittäjät tutkivat vaihtoehtoja ja valitsevat asianmukaisia ratkaisuja kyseisiin tilanteisiin. Tämän projektin tekoäly on lopulta melko yksinkertainen teknisesti, joten tilakoneen valinta osoittautui hyväksi ratkaisuksi.

Käytöspuut olisivat muuten erittäin hyvä vaihtoehto, mutta nämä toimivat parhaiten silloin, kun tekoälyllä on paljon erilaisia toimintoja ja tiloja. Tämän projektin tekoälyssä puolestaan on niin vähän eri tiloja, että tilakone soveltui paremmin toteutustarkoituksiin. Näillä perustein hyötypohjainen laskentakin olisi soveltunut toteutusvaihtoehdoksi. Tämän poissulkemi-

seen oli kuitenkin yksi olennainen syy. Tämä syy on se, että tekoälyn toimintaan haluttiin tilakohtaista siirtymäehtologiikkaa. Eli eri tiloissa on eri prioriteetit tilojen siirtymäehdoille. Tämä mahdollistaa paljon monimuotoisempien tekoälyjen toteutuksen. Hyötypohjaisessa toteutuksessa sen sijaan siirtymäehdot sijaitsevat yhdessä paikassa. Toki hyötypohjaiseen toteutukseen saa tehtyä omanlaisia järjestelmiä, mutta tilakone soveltui paljon paremmin tähän tarkoitukseen.

Kuten alun perin ajateltiinkin, niin erinäisiä ongelmia ilmeni projektin toteutuksen edetessä. Varsinaisen peliprojektin toteutus oli melko suoraviivaista, mutta tekoälyn toteutuksessa ilmeni kaikennäköisiä ongelmia. Kaikki ongelmatilanteet kuitenkin saatiin ratkottua loppujen lopuksi, joten minkäänlaisia projektin kannalta tuhoisia ongelmia ei esiintynyt onneksi. Olisiko näitä ongelmatilanteita voinut huomioida ennakkoon? Ohjelmiston kehitykselle on ominaista erinäisten ongelmatilanteiden ilmaantuminen ja virheenratkonta. Hyvä suunnittelutyö auttaa ehkäisemään ongelmatilanteiden syntyä, mutta täysin ongelmitta harvoin selviää peliprojektin toteutuksessa.

Loppujen lopuksi tämä opinnäytetyöprojekti on kokonaisuudessaan varsin onnistunut. Käytännöntyön suhteen ideana oli se, että aluksi tehdään peli perusmekaniikoiltaan valmiiksi. Tässä vaiheessa peli on alustavasti valmis tekoälyä lukuun ottamatta. Tämän jälkeen kehitettiin tekoäly, joka toimii näissä perusmekaniikoissa asiallisesti. Kun nämä asiat ovat valmiita, niin voidaan miettiä projektin jatkokehitystä. Pelimekaniikkoja ja tekoälyä voisi kehittää usealla eri tavalla paremmiksi, mutta tämän opinnäytetyöprojektin puitteissa ei enää aika riitä jatkokehitykseen. Tulevaisuus näyttää tullaanko tätä projektia jatkamaan eteenpäin vai ei.

LÄHTEET

KIRJALLISUUS

Buckland M. 2005. Programming Game AI by Example. Wordware Publishing Inc.

Millington I. 2006. Artificial Intelligence for Games. Morgan Kaufmann Publishers.

Scott B. 2002. Chapter 1.2. AI Game Programming Wisdom. Charles River Media.

Tozour P. 2002. Chapter 1.1. AI Game Programming Wisdom. Charles River Media.

INTERNET ARTIKKELIT

Boreal Games 2013. Understanding Component Entity Systems. Saatavilla: http://www.gamedev.net/page/resources/_/technical/game-programming/understanding-component-entity-systems-r3013 (Luettu 5.11.2013).

Graham R. 2003. Pathfinding in Computer Games http://gamesitb.com/pathgraham.pdf (Luettu 4.11.2013).

Hale R. 2009. 7 Ways to make your AI smarter http://www.agamesdesignblog.com/2009/03/09/7-ways-to-make-your-ai-smarter/ (Luettu 2.11.2013).

Kehoe D. 2013. Designing Artificial Intelligence for games (Part1) http://software.intel.com/en-us/articles/designing-artificial-intelligence-for-games-part-1 (Luettu 2.11.2013).

Knafla B. 2011. Introduction to Behavior Trees http://www.altdevblogaday.com/2011/02/24/introduction-to-behavior-trees/ (Luettu 3.11.2013).

Lester P. 2005. A* Pathfinding for Beginners http://www.gamedev.net/page/resources/_/technical/artificial-intelligence/a-pathfinding-for-beginners-r2003 (Luettu 4.11.2013).

Mark D. 2012. AI Architectures: A Culinary Guide
http://intrinsicalgorithm.com/IAonAI/2012/11/ai-architectures-a-culinary-guide-gdmag-article/ (Luettu 3.11.2013).

Mateas M. 2003. Expressive AI: Games and Artificial Intelligence
http://lmc.gatech.edu/~mateas/publications/MateasDIGRA2003.pdf (Luettu 2.11.2013).

Nareyek A. 2004. AI in computer games
http://www.ai-center.com/publications/nareyek-acmqueue04.pdf (Luettu 2.11.2013).

Qiao. Pathfinding Visual
http://qiao.github.io/PathFinding.js/visual/ (Luettu 4.11.2013).

TechTerms 2010. Artificial Intelligence
http://www.techterms.com/definition/artificial_intelligence (Luettu 2.11.2013).

Wexler J. 2002. Artificial Intelligence in Games
http://www.cs.rochester.edu/~brown/242/assts/termprojs/games.pdf (Luettu 2.11.2013).